coronagraph Documentation

Release 1.0

Jacob Lustig-Yaeger

Aug 30, 2019

CONTENTS

A Python noise model for directly imaging exoplanets with a coronagraph-equipped telescope. The original IDL code for this coronagraph model was developed and published by Tyler Robinson and collaborators (Robinson, Stapelfeldt & Marley 2016). This open-source Python version has been expanded upon in a few key ways, most notably, the `Telescope`, `Planet`, and `Star` objects used for reflected light coronagraph noise modeling can now be used for transmission and emission spectroscopy noise modeling, making this model a general purpose exoplanet noise model for many different types of observations.

To get started using the `coronagraph` noise model, take a look at the Quickstart tutorial and the examples. For more details about the full functionality of the code see the Application Programming Interface (API).

If you use this model in your own research please cite Robinson, Stapelfeldt & Marley (2016) and Lustig-Yaeger, Robinson & Arney (2019).

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

The simplest option is to install using `pip`:

```
pip install coronagraph
```

Users may also clone the repository on Github for the latest version of the code:

```
git clone https://github.com/jlustigy/coronagraph.git
cd coronagraph
python setup.py
```

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.2 Quickstart Tutorial

Let's begin by importing the coronagraph model and checking the version number.

[3]:
```
import coronagraph as cg
print(cg.__version__)
```

```
1.0
```

1.2.1 Model Inputs

This model uses Python objects to hold all of the specific telescope/coronagraph and astrophysical parameters that we want to assume for our noise calculations. For this simple example, we'll set up the `Telescope`, `Planet`, and `Star` by instantiating them with their default values.

[5]:
```
telescope = cg.Telescope()
print(telescope)
```

```
Coronagraph:
--------
- Telescope observing mode : IFS
- Minimum wavelength (um) : 0.3
- Maximum wavelength (um)  : 2.0
```

(continues on next page)

(continued from previous page)

```
- Spectral resolution (lambda / delta-lambda)  : 70.0
- Telescope/System temperature (K)  : 260.0
- Detector temperature (K)  : 50.0
- Telescope diameter (m)  : 8.0
- Telescope emissivity  : 0.9
- Inner Working Angle (lambda/D)  : 0.5
- Outer Working Angle (lambda/D)  : 30000.0
- Telescope throughput  : 0.2
- Raw Contrast  : 1e-10
- Dark current (s**-1)  : 0.0001
- Horizontal pixel spread of IFS spectrum  : 3.0
- Read noise per pixel  : 0.1
- Maximum exposure time (hr)  : 1.0
- Size of photometric aperture (lambda/D)  : 0.7
- Quantum efficiency  : 0.9
```

[6]:
```
planet = cg.Planet()
print(planet)
```

```
Planet:
-----
- Planet name  : earth
- Stellar type of planet host star : sun
- Distance to system (pc) : 10.0
- Number of exzodis (zodis) : 1.0
- Radius of planet (Earth Radii) : 1.0
- Semi-major axis (AU) : 1.0
- Phase angle (deg) : 90.0
- Lambertian phase function : 0.3183098861837907
- Zodiacal light surface brightness (mag/arcsec**2) : 23.0
- Exozodiacal light surface brightness (mag/arcsec**2) : 22.0
```

[7]:
```
star = cg.Star()
print(star)
```

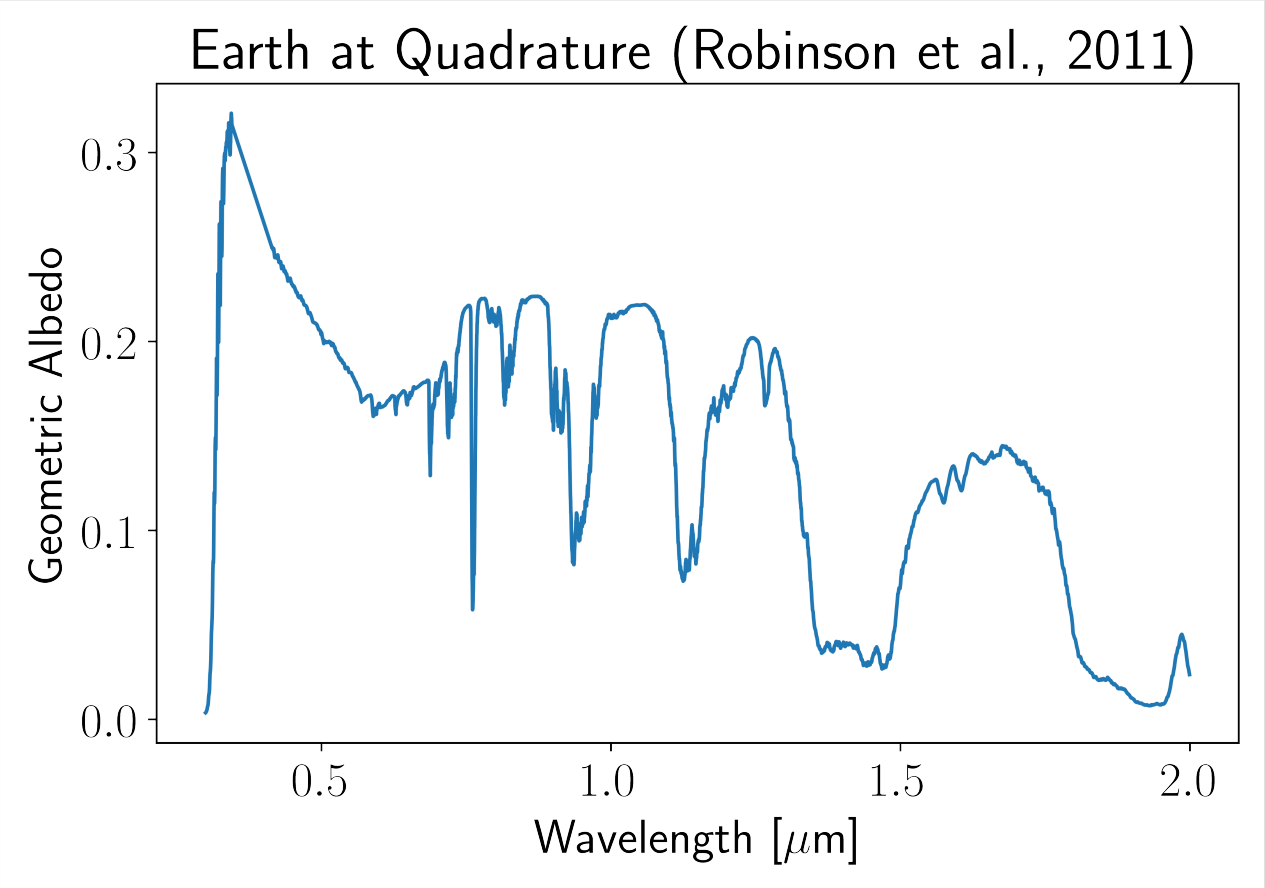
```
Star:
---
- Effective Temperature (K) : 5780.0
- Radius (Solar Radii) : 1.0
```

Now let's load in a high resolution model Earth geometric albedo spectrum from Robinson et al. (2011). This iconic spectrum is provided as part of the coronagraph model, but this step is usually project specific.

[8]:
```
# Load Earth albedo spectrum from Robinson et al. (2011)
lamhr, Ahr, fstar = cg.get_earth_reflect_spectrum()
```

We can take a look at the disk-integrated geomtric albedo spectrum of the Earth with realistic cloud coverage:

[9]:
```
# Create wavelength mask
m = (lamhr > telescope.lammin) & (lamhr < telescope.lammax)

# Plot
plt.plot(lamhr[m], Ahr[m])
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel("Geometric Albedo")
plt.title("Earth at Quadrature (Robinson et al., 2011)");
```



1.2.2 Running the coronagraph noise model

Now we want to calculate the photon count rates incident upon the detector due to the planet and noise sources. We can accomplish all of this with a `CoronagraphNoise` object, to which we pass our existing `telescope`, `planet`, and `star` configurations. In addition, we will set the nominal exposure time in hours, `texp`, and/or a desired signal-to-noise ratio to obtain in each spectral element, `wantsnr`. Note `texp` and `wantsnr` cannot both be satisfied simultaneously, and they can be changed at a later time without needing to recompute the photon count rates.

[10]:
```
# Define CoronagraphNoise object using our telescope, planet, and star
noise = cg.CoronagraphNoise(telescope = telescope,
                            planet = planet,
                            star = star,
                            texp = 10.0,
                            wantsnr = 10.0)
```

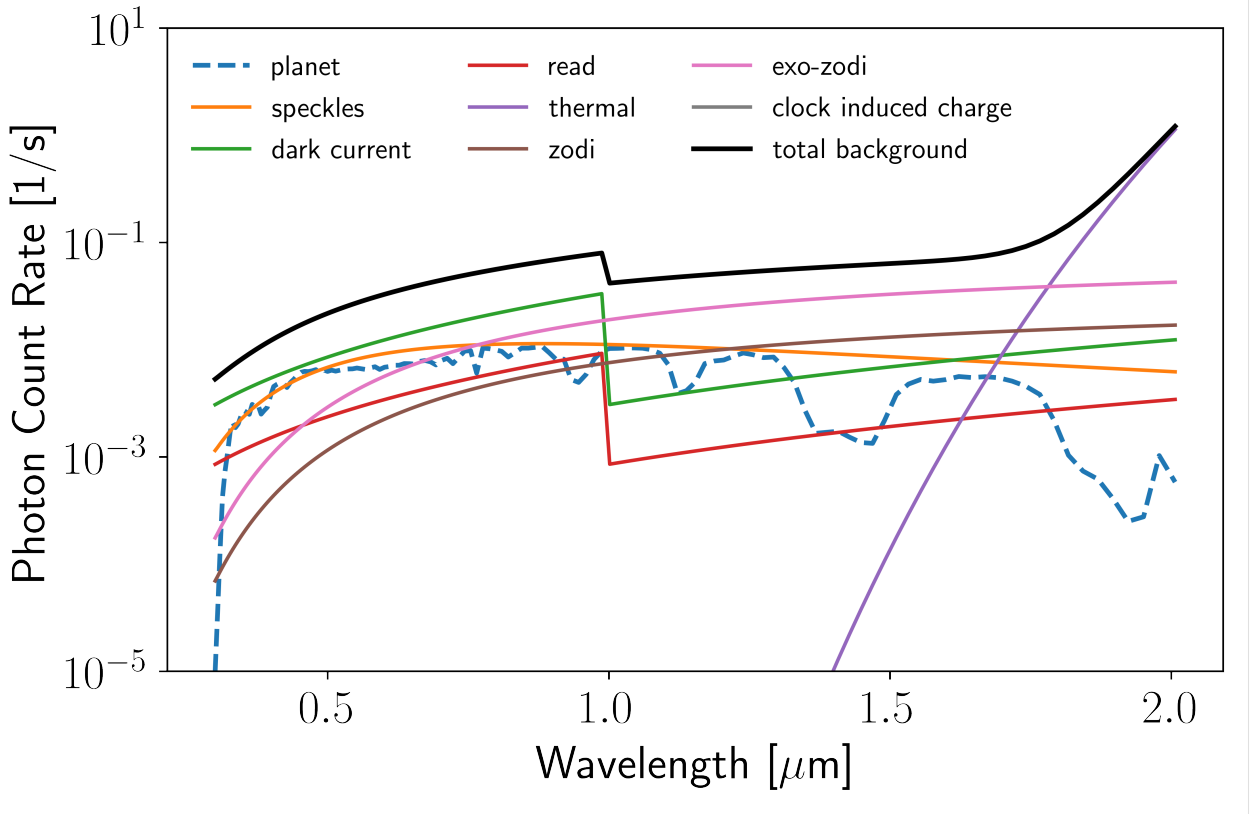
At this point we are ready to run the code to compute the desired photon count rates for our specific planetary spectrum. To do this simply call `run_count_rates` and provide the high-res planet spectrum, stellar spectrum, and wavelength grid.

[11]:
```
# Calculate the planet and noise photon count rates
noise.run_count_rates(Ahr, lamhr, fstar)
```

1.2.3 Analyzing results

After `run_count_rates` has been run, we can access the individual wavelength-dependent noise terms as attributes of the `noise` instance of the `CoronagraphNoise` object. For example:

[12]:
```
# Make plot
fig, ax = plt.subplots()
ax.set_xlabel("Wavelength [$\mu$m]")
ax.set_ylabel("Photon Count Rate [1/s]")

# Plot the different photon count rates
ax.plot(noise.lam, noise.cp, label = "planet", lw = 2.0, ls = "dashed")
ax.plot(noise.lam, noise.csp, label = "speckles")
ax.plot(noise.lam, noise.cD, label = "dark current")
ax.plot(noise.lam, noise.cR, label = "read")
ax.plot(noise.lam, noise.cth, label = "thermal")
ax.plot(noise.lam, noise.cz, label = "zodi")
ax.plot(noise.lam, noise.cez, label = "exo-zodi")
ax.plot(noise.lam, noise.cc, label = "clock induced charge")
ax.plot(noise.lam, noise.cb, label = "total background", lw = 2.0, color = "k")

# Tweak aesthetics
ax.set_yscale("log")
ax.set_ylim(bottom = 1e-5, top = 1e1)
ax.legend(fontsize = 12, framealpha = 0.0, ncol = 3);
```



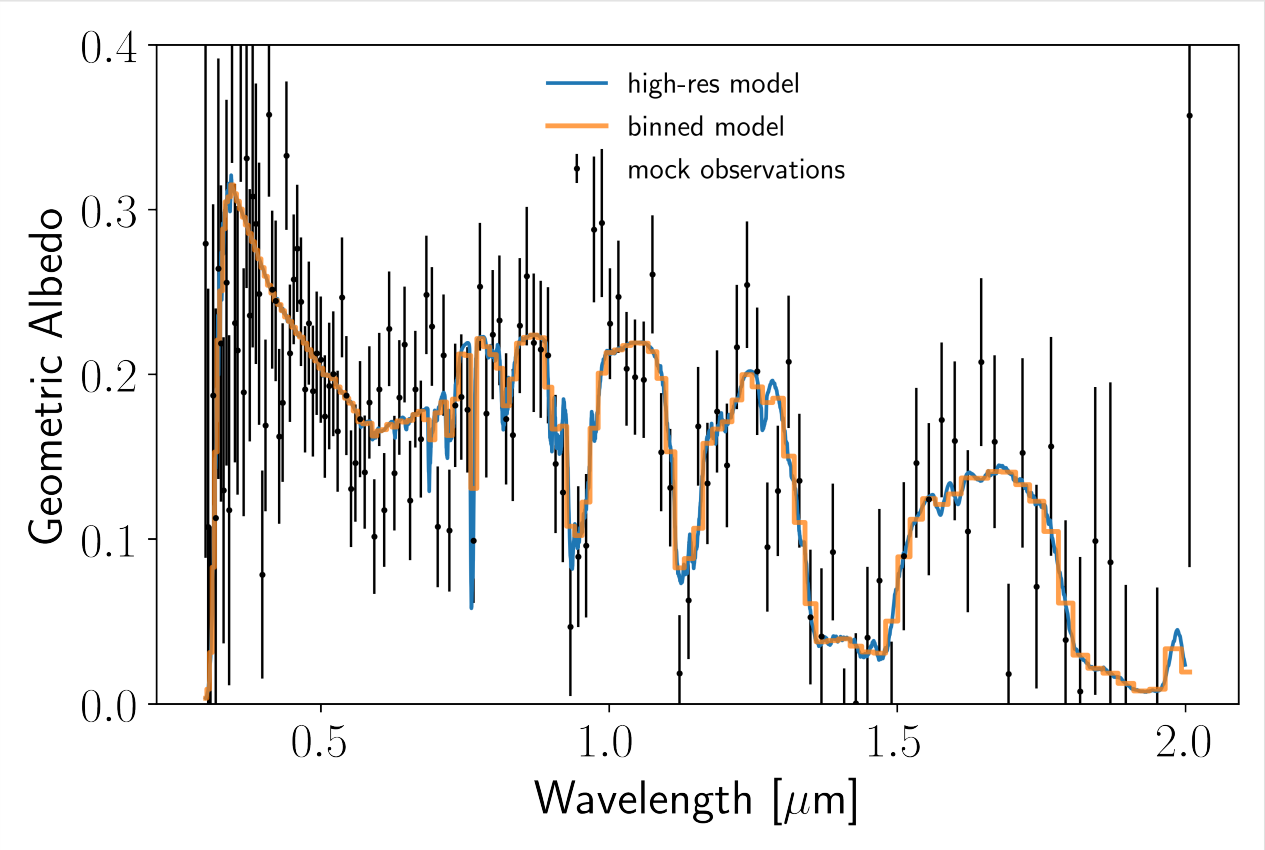
We can also plot the spectrum with randomly drawn Gaussian noise for our nominal exposure time:

[13]:
```
# Make plot
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots()
ax.set_xlabel("Wavelength [$\mu$m]")
ax.set_ylabel("Geometric Albedo")

# Plot the different photon count rates
ax.plot(lamhr[m], Ahr[m], label = "high-res model")
ax.plot(noise.lam, noise.A, label = "binned model", lw = 2.0, ls = "steps-mid", alpha␣
↪= 0.75)
ax.errorbar(noise.lam, noise.Aobs, yerr=noise.Asig, fmt = ".", c = "k", ms = 3, lw =␣
↪1.0, label = "mock observations")

# Tweak aesthetics
ax.set_ylim(bottom = 0.0, top = 0.4)
ax.legend(fontsize = 12, framealpha = 0.0);
```



A version of the above plot can be made using `noise.plot_spectrum()`, but it is not used here for model clarity in this tutorial.

The above plot gives us a quick look at the data quality that we might expect for an observation using the telescope and system setup. These data can be saved and used in retrieval tests to see if the true underlying atmospheric structure of the Earth can be extracted.

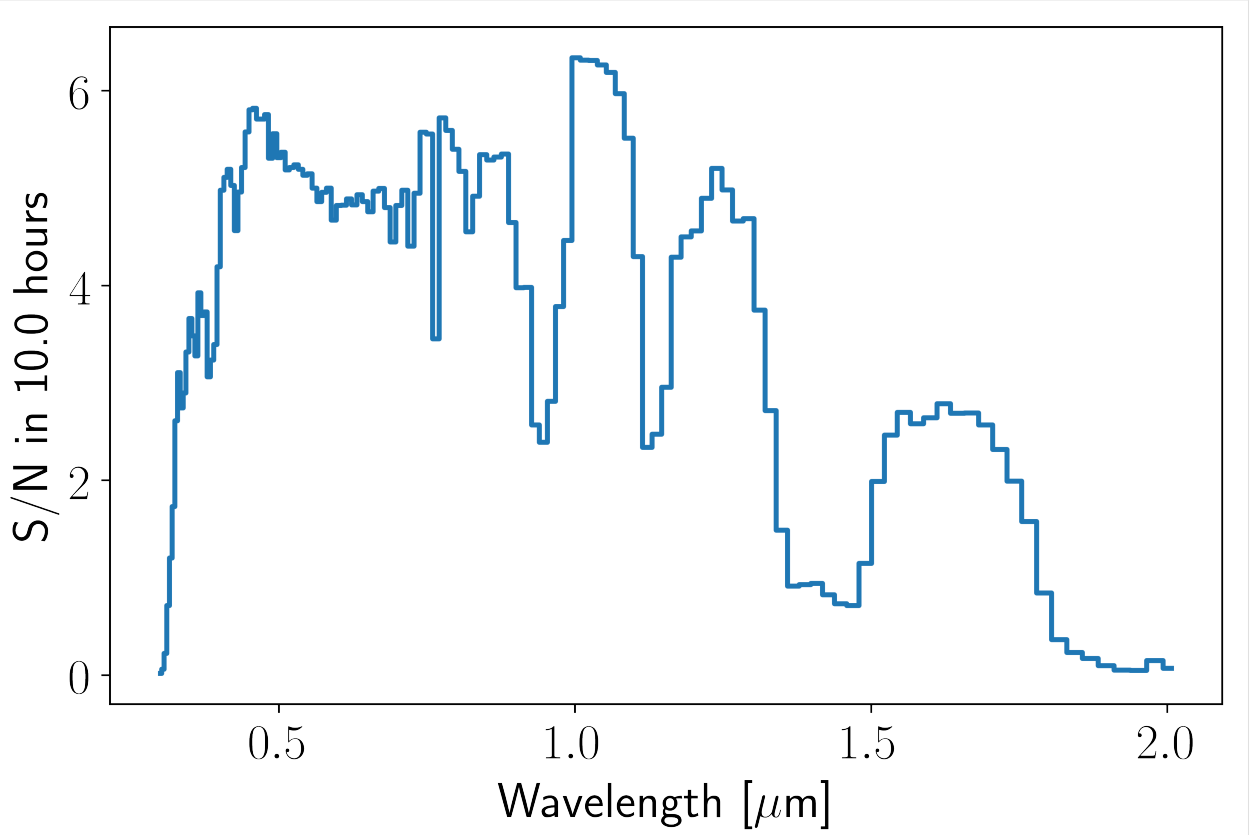
It's also useful to look at the signal-to-noise (S/N) ratio in each spectral element to see what wavelengths we are getting the highest S/N. We can access the wavelength dependent S/N for our selected exposure time via `noise.SNRt`. For example,

[14]:
```
# Make plot
fig, ax = plt.subplots()
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel("Wavelength [$\mu$m]")
ax.set_ylabel("S/N in %.1f hours" %noise.wantsnr)

# Plot the different photon count rates
ax.plot(noise.lam, noise.SNRt, lw = 2.0, ls = "steps-mid");
```

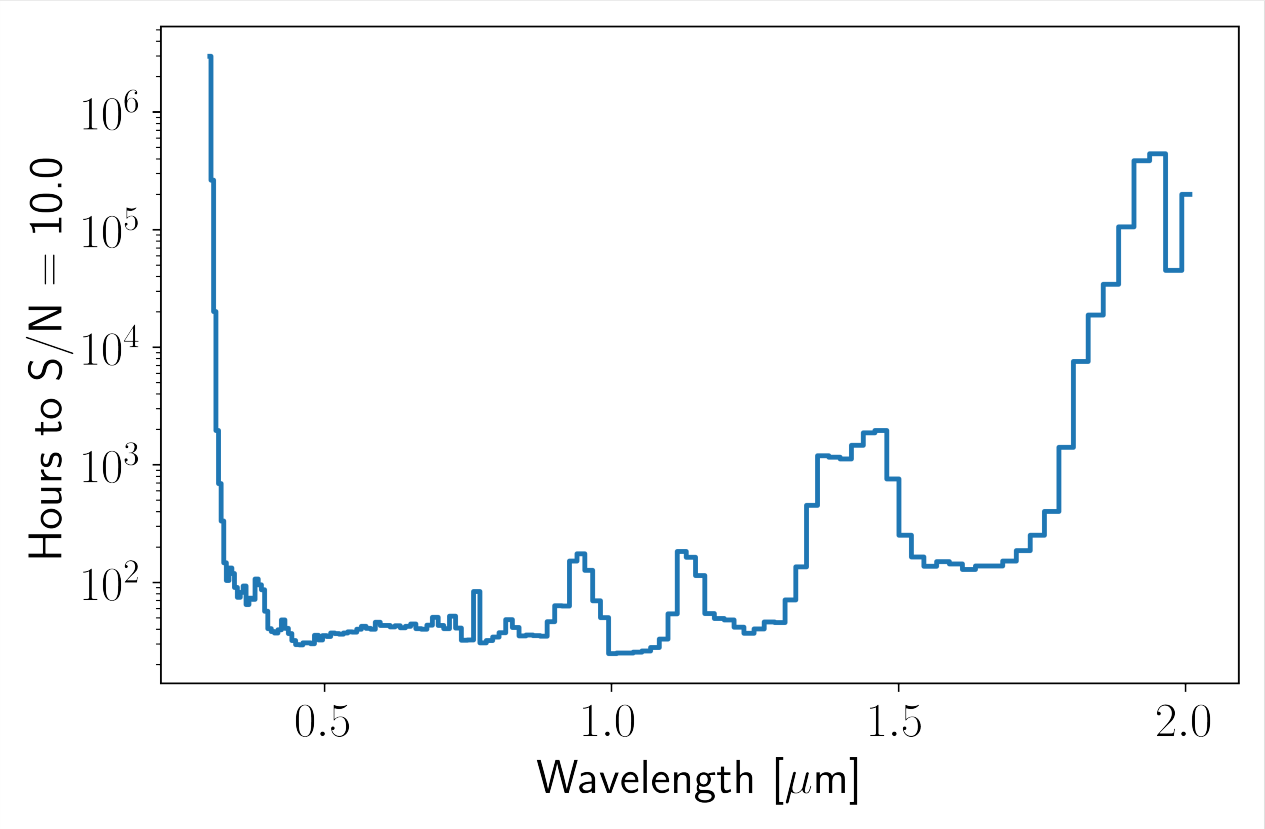


A version of the above plot can be made using `noise.plot_SNR()`.

We can see that the S/N has the same general shape as the stellar SED. We are, after all, observing stellar light reflected off a planet. As a result, the S/N is the highest where the Sun outputs the most photons: around 500 nm. But the S/N is not just the stellar SED, it is the stellar SED *convolved with the planet's reflectivity*. So we get lower S/N in the bottom of molecular absorption bands and higher S/N in the continuum between the bands. The peak in S/N near 0.75 µm is due to the increase in albedo of the Earth's surface (particularly land and vegetation) at that wavelength compared to shorter wavelengths where the Sun emits more photons.

Finally, let's take a look at the exposure time necessary to achieve a given signal-to-noise ratio (`wantsnr`). This calculated quantity can be accessed via `noise.DtSNR`. For example,

[15]:
```
# Make plot
fig, ax = plt.subplots()
ax.set_xlabel("Wavelength [$\mu$m]")
ax.set_ylabel("Hours to S/N = %.1f" %noise.texp)

# Plot the different photon count rates
ax.plot(noise.lam, noise.DtSNR, lw = 2.0, ls = "steps-mid")

# Tweak aesthetics
ax.set_yscale("log");
```



A version of the above plot can be made using `noise.plot_time_to_wantsnr()`.

The exposure time in each spectral element is inversely proportional to the S/N squared, $t_{\rm exp} \propto ({\rm S/N})^{-2}$, so this plot is inverted compared to the last one. Here we see that ridiculously infeasible exposure times are required to achieve high S/N at the wavelengths where fewer photons are reflected off the planet. Of course, this is all assuming that the exoplanet we are looking at *is* Earth. This is an extremely useful benchmark calculation to make (let's make sure we build a telescope capable of studying an exact exo-Earth), but as we see in this example, the spectrum, exposure time, and S/N are all quite dependent on the nature of the exoplanet, which we won't know *a priori*. So now you can use this model with your own simulated exoplanet spectra to see what a future telescope can do for you!

1.3 Examples

A collection of Jupyter Notebooks that demonstrate how to use `coronagraph`.

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.3.1 Degrading a Spectrum to Lower Resolution

[3]:
```
import coronagraph as cg
print(cg.__version__)
```

```
1.0
```

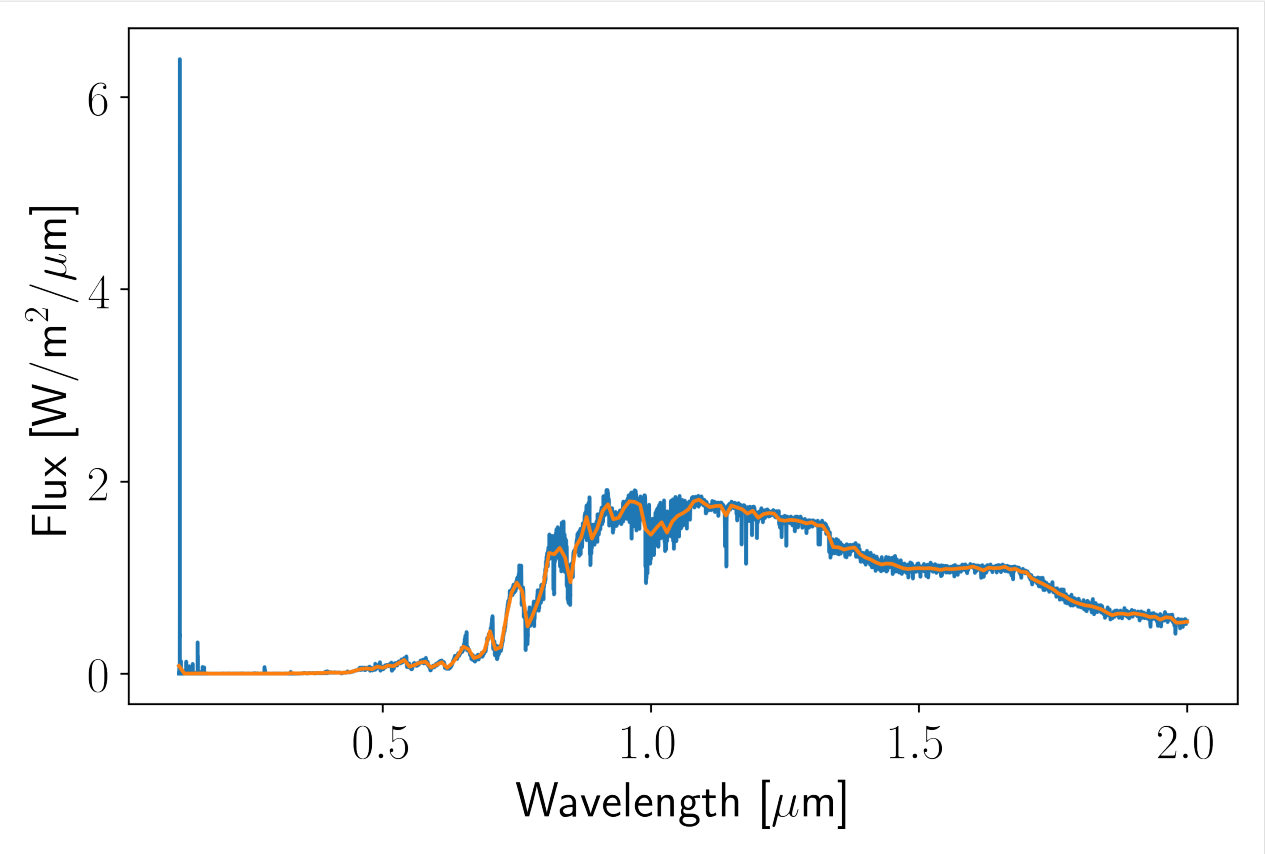
Proxima Centauri Stellar Spectrum

First, let's grab the stellar spectral energy distribution (SED) of Proxima Centauri from the VPL website:

[4]:
```
# The file is located at the following URL:
url = "http://vpl.astro.washington.edu/spectra/stellar/proxima_cen_sed.txt"

# We can open the URL with the following
import urllib.request
response = urllib.request.urlopen(url)

# Read file lines
data = response.readlines()

# Remove header and extraneous info
tmp = np.array([np.array(str(d).split("\\")) for d in data[25:]])[:,[1,2]]

# Extract columns
lam = np.array([float(d[1:]) for d in tmp[:,0]])
flux = np.array([float(d[1:]) for d in tmp[:,1]])
```

Now let's set the min and max for our low res wavelength grid, use `construct_lam` to create the low-res wavelength grid, use `downbin_spec` to make a low-res spectrum, and plot it.

[5]:
```
# Set the wavelength and resolution parameters
lammin = 0.12
lammax = 2.0
R = 200
dl = 0.01

# Construct new low-res wavelength grid
wl, dwl = cg.noise_routines.construct_lam(lammin, lammax, dlam = dl)

# Down-bin flux to low-res
flr = cg.downbin_spec(flux, lam, wl, dlam=dwl)

# Plot
m = (lam > lammin) & (lam < lammax)
plt.plot(lam[m], flux[m])
plt.plot(wl, flr)
#plt.yscale("log")
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel(r"Flux [W/m$^2$/$\mu$m]");
```



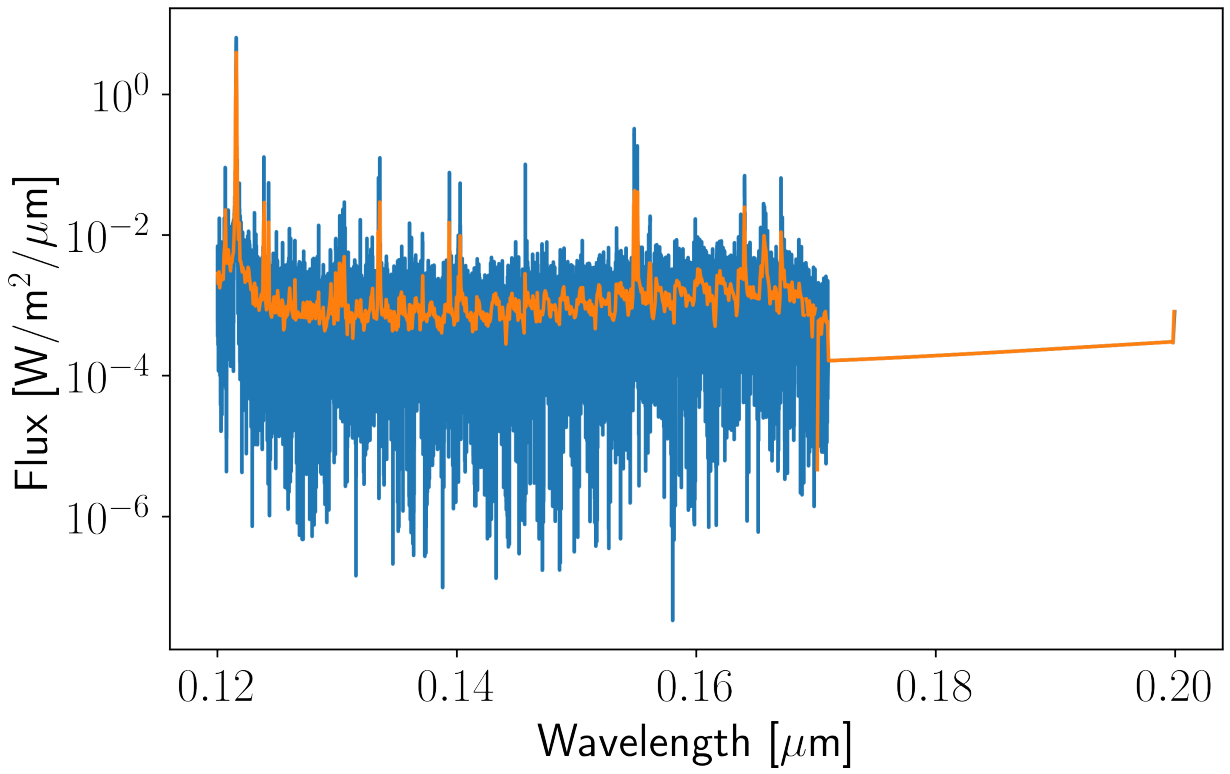
NaNs can occur when no high-resolution values exist within a given lower-resolution bein. How many NaNs are there?

[6]:
```
print(np.sum(~np.isfinite(flr)))
```

```
0
```

Let's try it again now focusing on the UV with a higher resolution.

[7]:
```
# Set the wavelength and resolution parameters
lammin = 0.12
lammax = 0.2
R = 2000

# Construct new low-res wavelength grid
wl, dwl = cg.noise_routines.construct_lam(lammin, lammax, R)

# Down-bin flux to low-res
flr = cg.downbin_spec(flux, lam, wl, dlam=dwl)

# Plot
m = (lam > lammin) & (lam < lammax)
plt.plot(lam[m], flux[m])
plt.plot(wl, flr)
plt.yscale("log")
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel(r"Flux [W/m$^2$/$\mu$m]");
```



[8]:
```
print(np.sum(~np.isfinite(flr)))
```

```
26
```

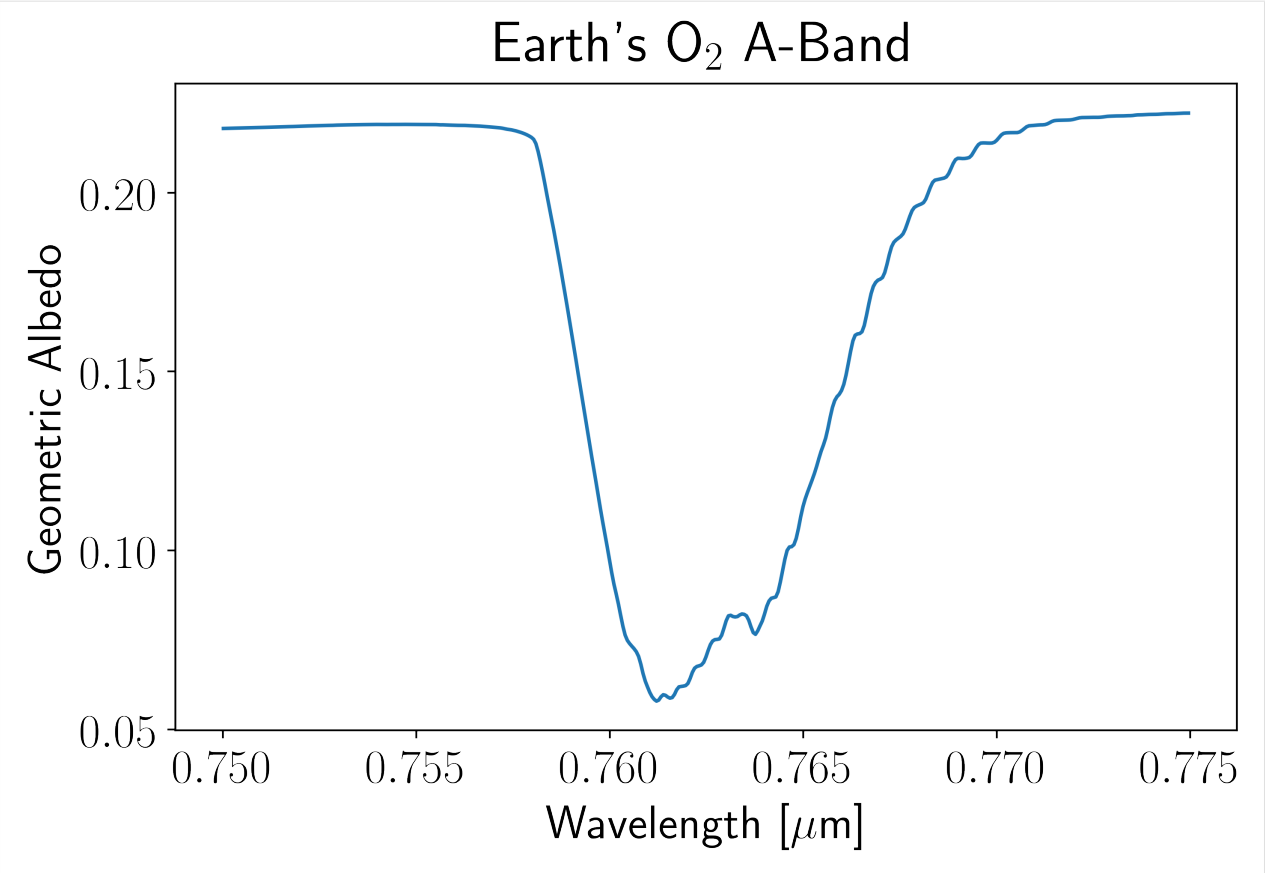
Optimal Resolution for Observing Earth's O_2 A-band

Let's load in the Earth's reflectance spectrum (Robinson et al., 2011).

[4]:
```
lamhr, Ahr, fstar = cg.get_earth_reflect_spectrum()
```

Now let's isolate just the O_2 A-band.

[5]:
```
lammin = 0.750
lammax = 0.775

# Create a wavelength mask
m = (lamhr > lammin) & (lamhr < lammax)

# Plot the band
plt.plot(lamhr[m], Ahr[m])
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel("Geometric Albedo")
plt.title(r"Earth's O$_2$ A-Band");
```



Define a set of resolving powers for us to loop over.

[6]:
```
R = np.array([1, 10, 30, 70, 100, 150, 200, 500, 1000])
```

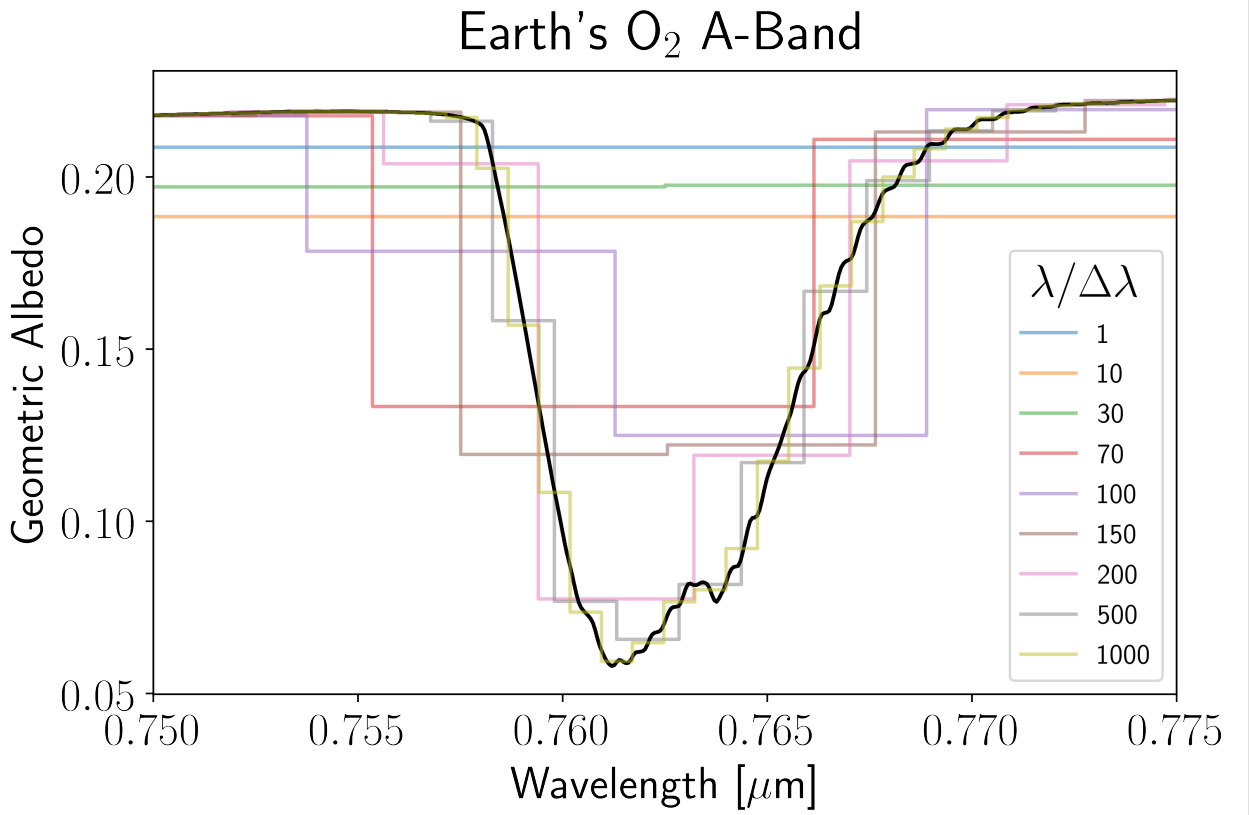
Let's down-bin the high-res spectrum at each `R`. For each `R` in the loop we will construct a new wavelength grid, down-bin the high-res spectrum, and plot the degraded spectrum. Let's also save the minimum value in the degraded spectrum to assess how close we get to the actual bottom of the band.

[7]:
```
bottom_val = np.zeros(len(R))

# Loop over R
for i, r in enumerate(R):

    # Construct new low-res wavelength grid
    wl, dwl = cg.noise_routines.construct_lam(lammin, lammax, r)

    # Down-bin flux to low-res
    Alr = cg.downbin_spec(Ahr, lamhr, wl, dlam=dwl)

    # Plot
    plt.plot(wl, Alr, ls = "steps-mid", alpha = 0.5, label = "%i" %r)

    # Save bottom value
    bottom_val[i] = np.min(Alr)

# Finsh plot
plt.plot(lamhr[m], Ahr[m], c = "k")
```

(continues on next page)

(continued from previous page)

```
plt.xlim(lammin, lammax)
plt.legend(fontsize = 12, title = r"$\lambda / \Delta \lambda$")
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel("Geometric Albedo")
plt.title(r"Earth's O$_2$ A-Band");
```



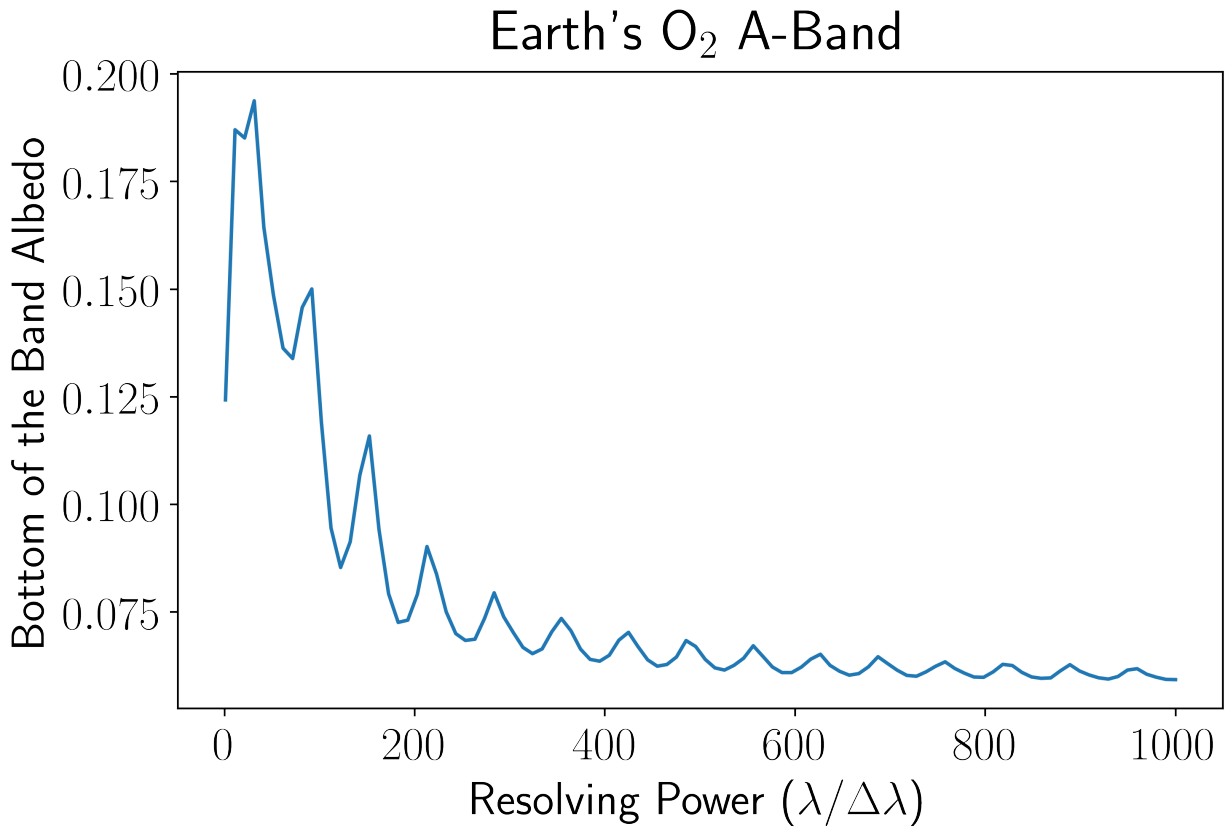
We can now compare the bottom value in low-res spectra to the bottom of the high-res spectrum.

[68]:
```
# Create resolution array to loop over
Nres = 100
R = np.linspace(1,1000, Nres)

# Array to store bottom-of-band albedos
bottom_val = np.zeros(len(R))

# Loop over R
for i, r in enumerate(R):

    # Construct new low-res wavelength grid
    wl, dwl = cg.noise_routines.construct_lam(lammin, lammax, r)

    # Down-bin flux to low-res
    Alr = cg.downbin_spec(Ahr, lamhr, wl, dlam=dwl)

    # Save bottom value
    bottom_val[i] = np.min(Alr)

# Make plot
```

(continues on next page)

(continued from previous page)

```
plt.plot(R, bottom_val);
plt.xlabel(r"Resolving Power ($\lambda / \Delta \lambda$)")
plt.ylabel("Bottom of the Band Albedo")
plt.title(r"Earth's O$_2$ A-Band");
```



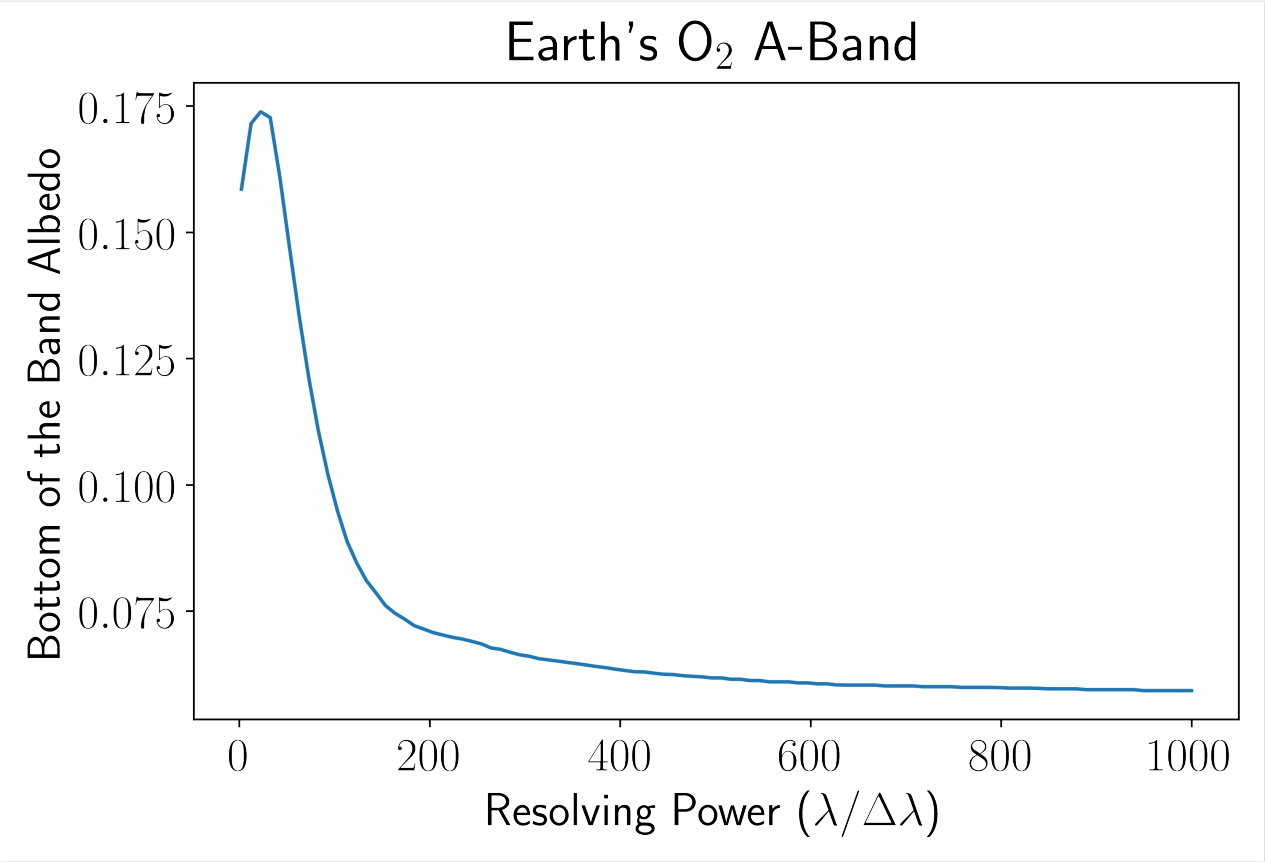
The oscillations in the above plot are the result of non-optimal placement of the resolution element relative to the oxygen A-band central wavelength. We can do better than this by iterating over different minimum wavelength positions.

[183]:
```
# Create resolution array to loop over
Nres = 100
R = np.linspace(2,1000, Nres)

# Set number of initial positions
Ntest = 20

# Arrays to save quantities
bottom_vals = np.nan*np.zeros([len(R), Ntest])
best = np.nan*np.zeros(len(R), dtype=int)
Alrs = []
lams = []

# Loop over R
for i, r in enumerate(R):

    # Set grid of minimum wavelengths to iterate over
    lammin_vals = np.linspace(lammin - 1.0*0.76/r, lammin, Ntest)
```

(continues on next page)

(continued from previous page)

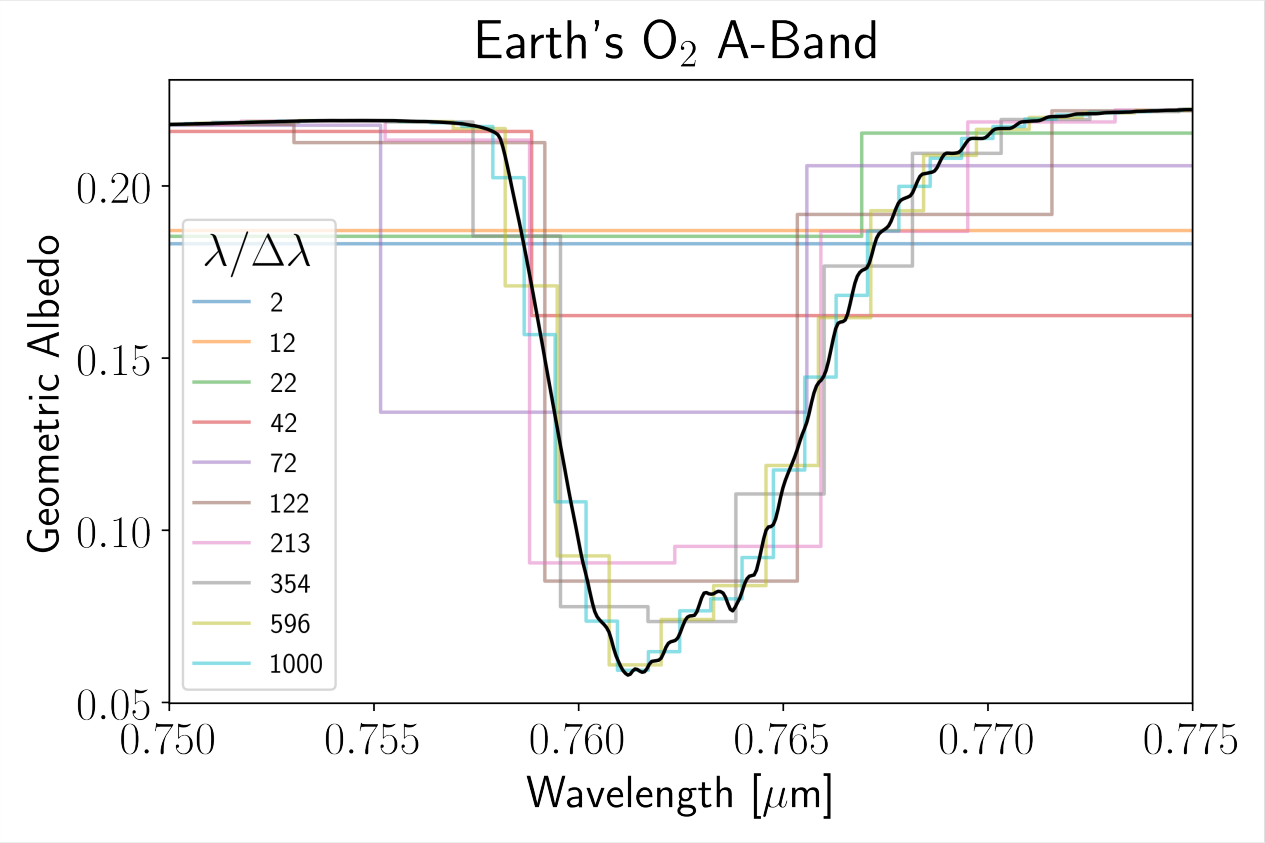
```
    # Loop over minimum wavelengths to adjust bin centers
    for j, lmin in enumerate(lammin_vals):

        # Construct new low-res wavelength grid
        wl, dwl = cg.noise_routines.construct_lam(lmin, lammax, r)

        # Down-bin flux to low-res
        Alr = cg.downbin_spec(Ahr, lamhr, wl, dlam=dwl)

        # Keep track of the minimum
        if ~np.isfinite(best[i]) or (np.nansum(np.min(Alr) < bottom_vals[i,:]) > 0):
            best[i] = j

        # Save quantities
        bottom_vals[i,j] = np.min(Alr)
        Alrs.append(Alr)
        lams.append(wl)

# Reshape saved arrays
Alrs = np.array(Alrs).reshape((Ntest, Nres), order = 'F')
lams = np.array(lams).reshape((Ntest, Nres), order = 'F')
best =  np.array(best, dtype=int)

# Plot the global minimum
plt.plot(R, np.min(bottom_vals, axis = 1));
plt.xlabel(r"Resolving Power ($\lambda / \Delta \lambda$)")
plt.ylabel("Bottom of the Band Albedo")
plt.title(r"Earth's O$_2$ A-Band");
```



In the above plot we are looking at the minimum albedo of the oxygen A-band *after optimizing the central band location*, and we can see that the pesky oscillations are now gone. At low resolution the above curve quickly decays as less and less continuum is mixed into the band measurement, while at high resolution the curve asymptotes to the true minimum of the band.

Essentially what we have done is ensure that, when possible, the oxygen band is not split between multiple spectral elements. This can be seen in the following plot, which samples a few of the optimal solutions. It doesn't look too different from the first version of this plot, but it does more efficiently sample the band shape (note that the lower resolution spectra have deeper bottoms and more tightly capture the band wings).

[186]:
```
# Get some log-spaced indices so we're not plotting all R
iz = np.array(np.logspace(np.log10(1), np.log10(100), 10).round(), dtype=int) - 1

# Loop over R
for i, r in enumerate(R):

    # Plot some of the resolutions
    if i in iz:
        plt.plot(b[best[i], i], a[best[i], i], ls = "steps-mid", alpha = 0.5, label =
↪"%i" %r)
        plt.xlim(lammin, lammax)

# Finsh plot
plt.plot(lamhr[m], Ahr[m], c = "k")
plt.legend(fontsize = 12, title = r"$\lambda / \Delta \lambda$")
plt.xlabel(r"Wavelength [$\mu$m]")
plt.ylabel("Geometric Albedo")
plt.title(r"Earth's O$_2$ A-Band");
```



As you can see, much can be done with the simple re-binning functions, but the true utility of the `coronagraph` model comes from the noise calculations. Please refer to the other tutorials and examples for more details!

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.3.2 Secondary Eclipse Spectroscopy

The `coronagraph` model can be used to model exoplanet observations using telescopes that don't even have a coronagraph. Here we will walk through a few examples of how to simulate secondary eclipse spectroscopy using the generic telescope noise modeling tools available with the `coronagraph` package.

[3]:
```
import coronagraph as cg
print(cg.__version__)
```

```
1.0
```

Earth in Emission

Let's start out by loading a model spectrum of the Earth:

[4]:
```
lam, tdepth, fplan, fstar = cg.get_earth_trans_spectrum()
```

Now we'll define the telescope/instrument parameters for our simulation. Let's go big for this demo and use a 15 meter space telescope, with 50% throughput, and a mirror temperature at absolute zero so there is no thermal noise.

[5]:
```
telescope = cg.Telescope(Tput = 0.5,       # Throughput
                         D = 15.,          # Diameter [m]
                         R = 70,           # Resolving power (lam / dlam)
                         lammin  = 5.0,    # Minimum Wavelength [um]
                         lammax  = 20.0,   # Maximum Wavelength [um]
                         Tsys = 0.0,       # Telescope mirror temperature [K]
                        )
```

We'll define the observed system as an Earth-Sun analog at 10 pc.

[6]:
```
planet = cg.Planet(a = 1.0,    # Semi-major axis [AU]
                   d = 10.0,   # Distance [pc]
                   Rp = 1.0    # Planet Radius [Earth Radii]
                  )

star = cg.Star(Rs = 1.0,       # Stellar Radius [Solar Radii]
               Teff = 5700.    # Stellar Effective Temperature [K]
              )
```

Now let's specify the transit/eclipse duration `tdur` for the planet (Earth is this case), the number of eclipses to observe `ntran`, and the amount of observing out-of-eclipse (in units of eclipse durations) for each secondary eclipse observation `nout`. We're going to simulate the spectrum we would get after observing 1000 secondary eclipses of the Earth passing behind the Sun. Obviously this is ridiculous. One thousand secondary eclipses is not only a minimum of 8000 hours of observational time, but it would take 1000 years to aquire such a dataset! I promise that there is a lesson to be leared here, so let's push forward into infeasibility.

[17]:
```
tdur = 8.0 * 60 * 60  # Transit/Eclipse duration [seconds]
ntran = 1e3           # Number of eclipses
nout = 2.0            # Number of out-of-eclipse durations [transit durations]
wantsnr = 10.0        # Desired S/N per resolution element (when applicable)
```

Now, we're ready to instantiate an `EclipseNoise` object for our simulation, which contains all of the information needed to perform the noise calculation.

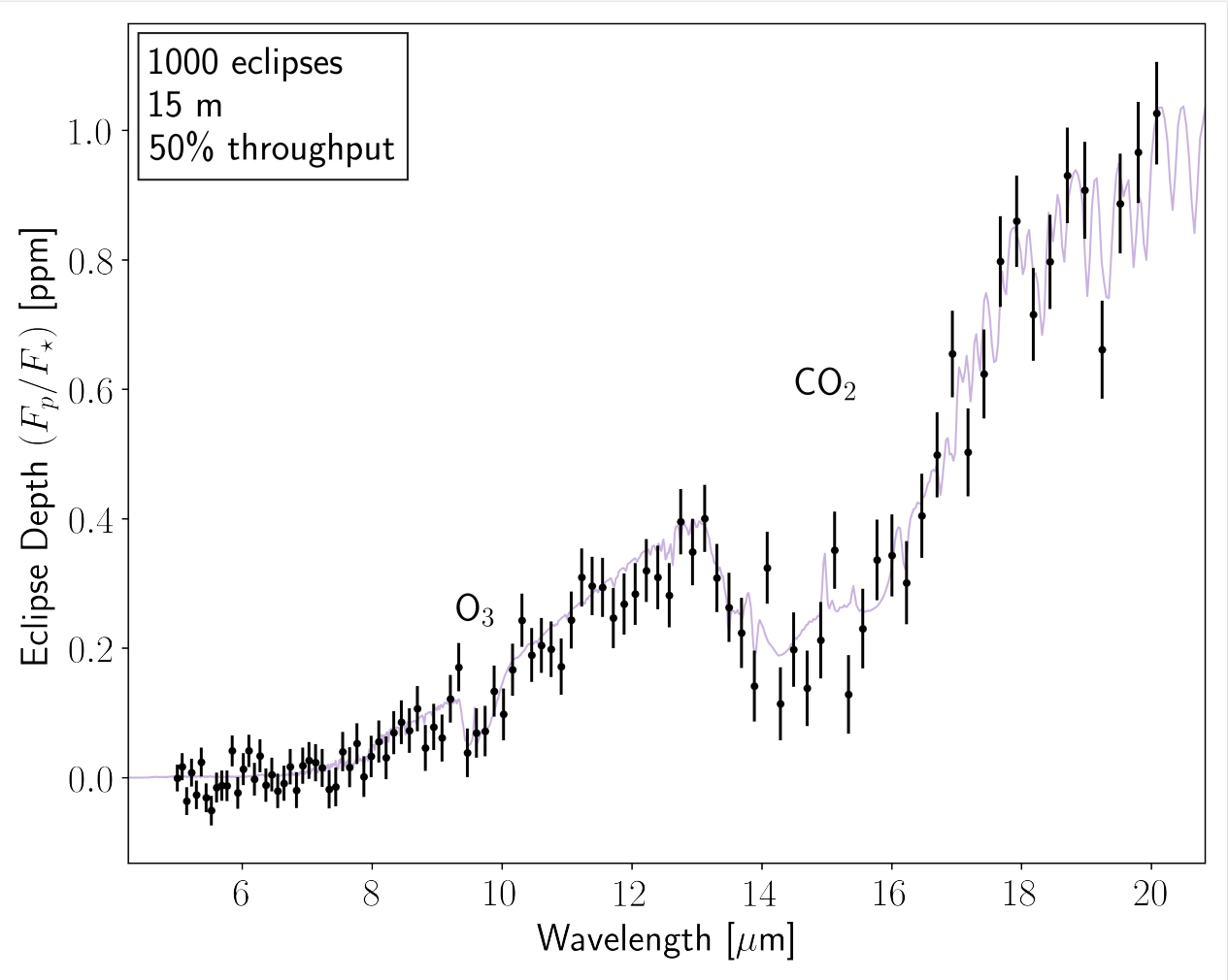
[18]:
```
en = cg.EclipseNoise(tdur = tdur,             # Transit Duration
                     telescope = telescope,   # Telescope object
                     planet = planet,         # Planet object
                     star = star,             # Star object
                     ntran = ntran,           # Number of eclipses to observe
                     nout = nout,             # Number of out-of-eclipse observing
                     wantsnr = wantsnr)       # Desired S/N per resolution element␣
↪(when applicable)
```

At this point we are ready to run the simulation, so we simply call the `run_count_rates` method:

[19]:
```
en.run_count_rates(lam, fplan, fstar)
```

Let's now plot our fiducial secondary eclipse spectrum.

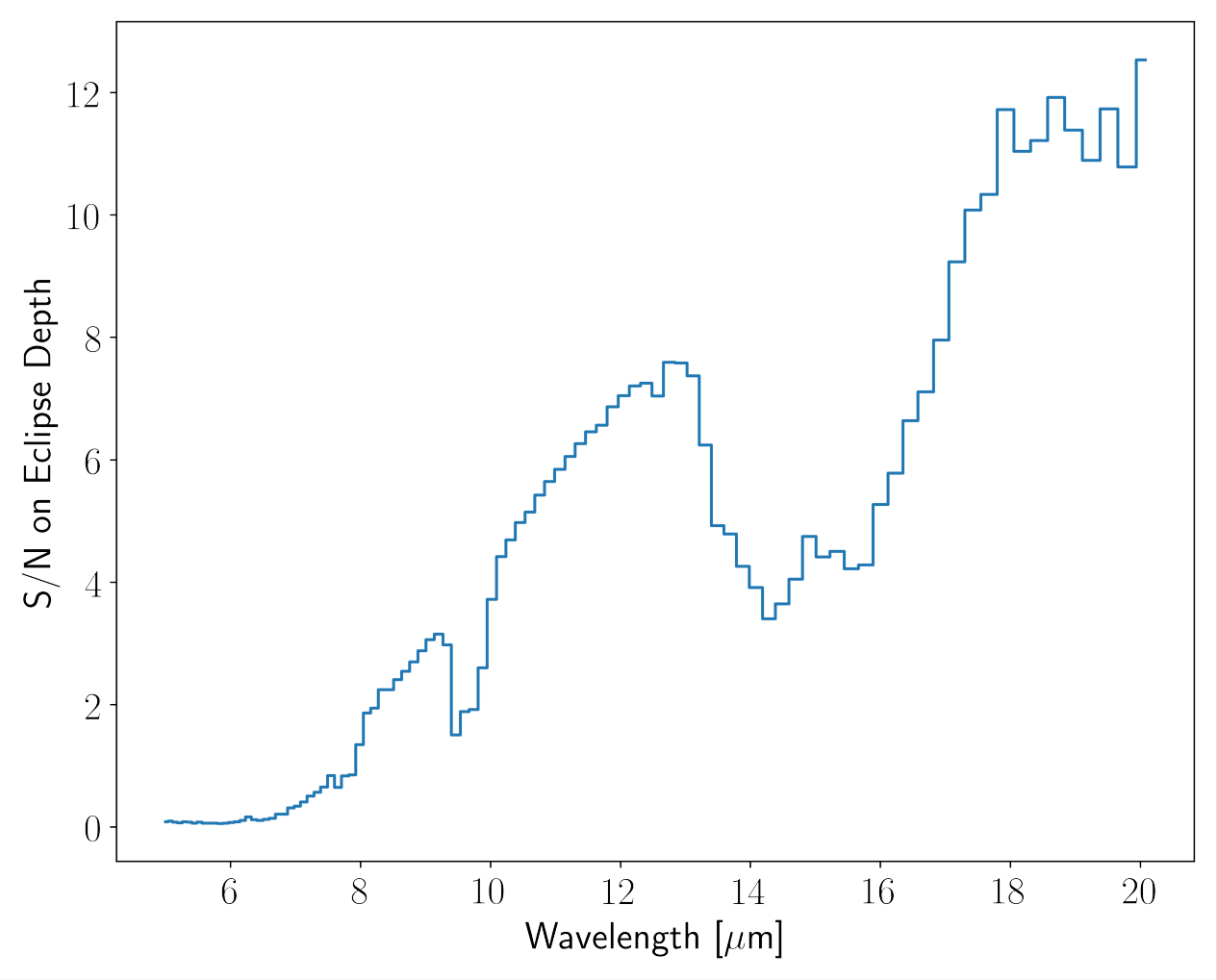
[20]:
```
fig, ax = en.plot_spectrum(SNR_threshold=0.0, Nsig=None)
ax.text(15.0, 0.6, r"CO$_2$", va = "center", ha = "center");
ax.text(9.6, 0.25, r"O$_3$", va = "center", ha = "center");
```



We can three key features. First, we note the general rising trend in the planet-to-star flux contrast with increasing wavelength. This occurs because, even though the Earth is much *much* more faint than the Sun at visible wavelengths, the blackbody intensity is weakly dependent on temperature in the Rayleigh-Jeans limit, so at these longer wavelengths the 288 K Earth and 5700 K Sun have more similar intensities. Second, we see absorption due to CO_2 at 15 μm. Finally, we see absorption due to O_3 at 9.6 μm. The rise in planet-to-star flux contrast with wavelength increases the relative size of absorption features in the emission spectrum. At first blush this would appear to make detecting molecules easier at longer wavelengths, however, photons are few and far between at these wavelengths and thermal noise tends to increase with wavelength, making the optimal wavelengths for molecular detection more complicated and telescope-dependent.

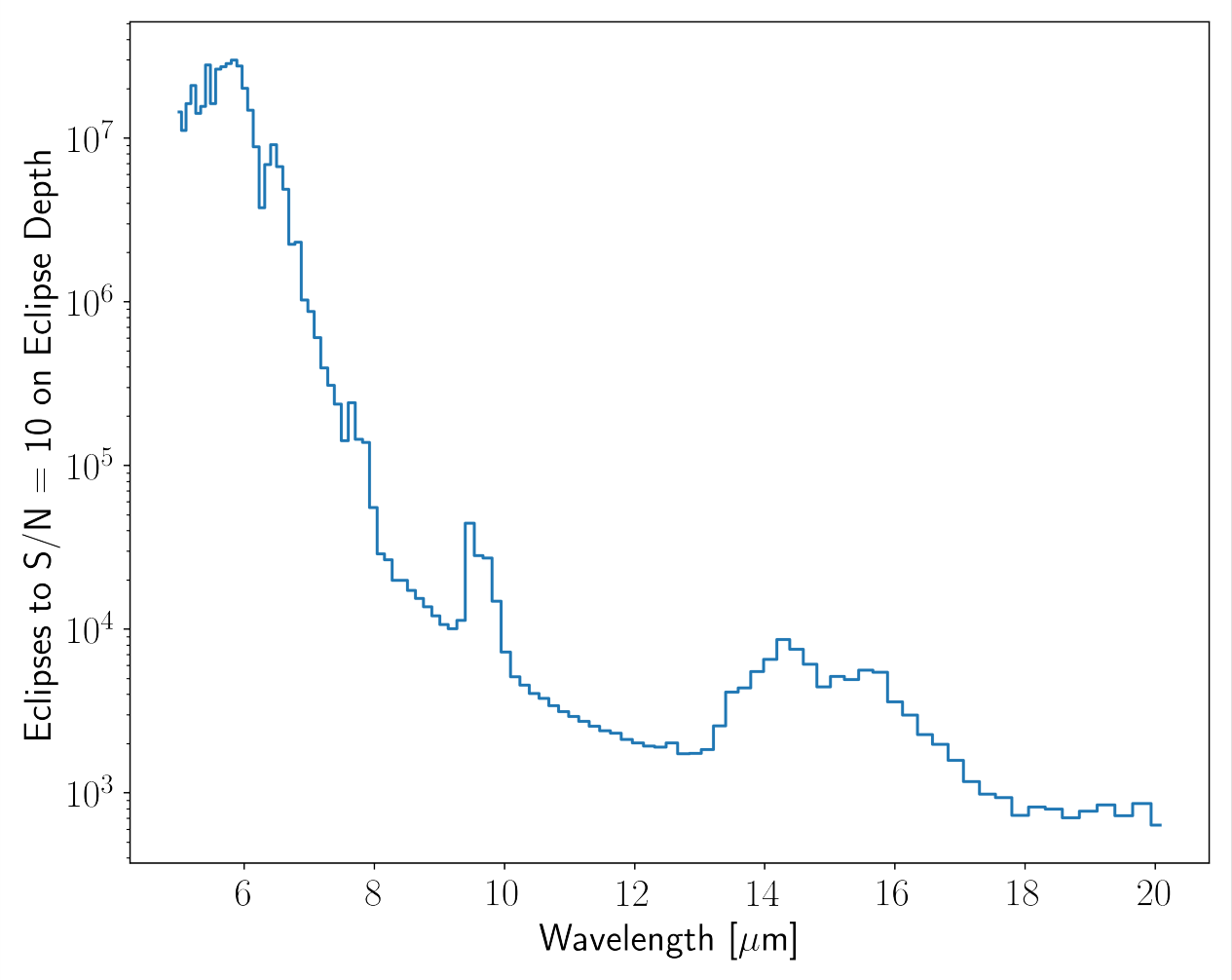
Let's take a look that the signal-to-noise as a function of wavelength.

[21]:
```
fig, ax = en.plot_SNRn()
```



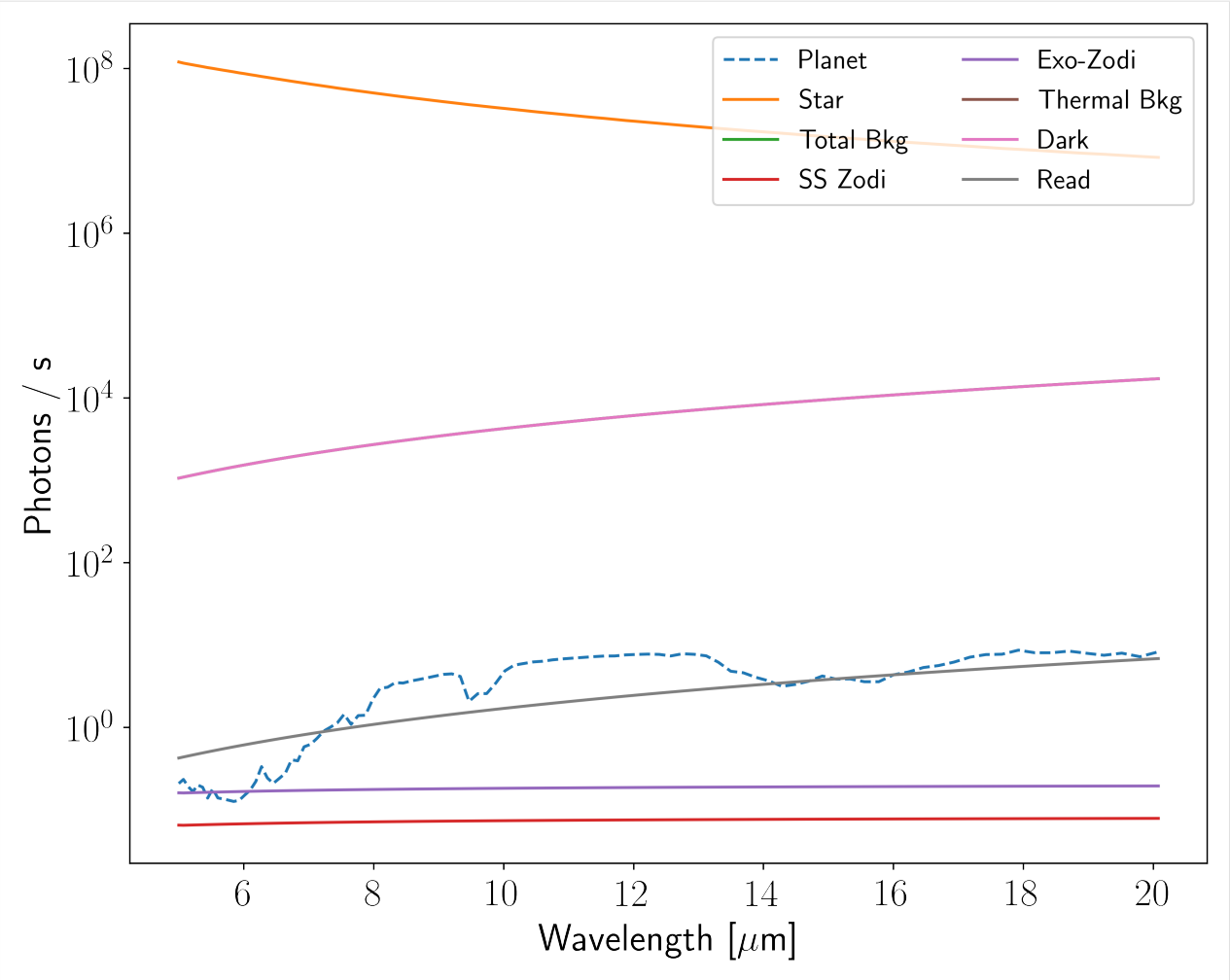
Sometimes it's nice to see the number of eclipses that must be observed to get a given S/N in each spectral element. Let's take a look at these devastating numbers for the Earth around the Sun.

[22]:
```
fig, ax = en.plot_ntran_to_wantsnr()
```



So this is really just too ridiculous. The ultimate reason for that can be gleaned from the last plot in this series, where we take a look at the photon count rates incident upon the detector from various noise sources compared to our signal.

[23]:
```
fig, ax = en.plot_count_rates()
```



We see that it's all about the star. The themral emission from the Sun outshines the thermal emission from the Earth by ~6 orders of magnitude at 20 μm, and there aren't enough photons to overcome this contrast in any reasonable amount of time.

Simulating a *featureless* spectrum

We've heard a lot about featureless *transmission* spectra. This can occur due to high altitude aerosols or heavy and/or cool atmospheres. But what does a featureless emission spectrum look like?

In short, it looks like a ratio of blackbody fluxes:

$$\frac{F_p}{F_\star} \approx \frac{B_p}{B_\star}\left(\frac{R_p}{R_\star}\right)^2$$

But let's see how we can investigate this with the `coronagraph` model.

We're going to use the same planet and star as above, but now let's set a temperature for the planet:

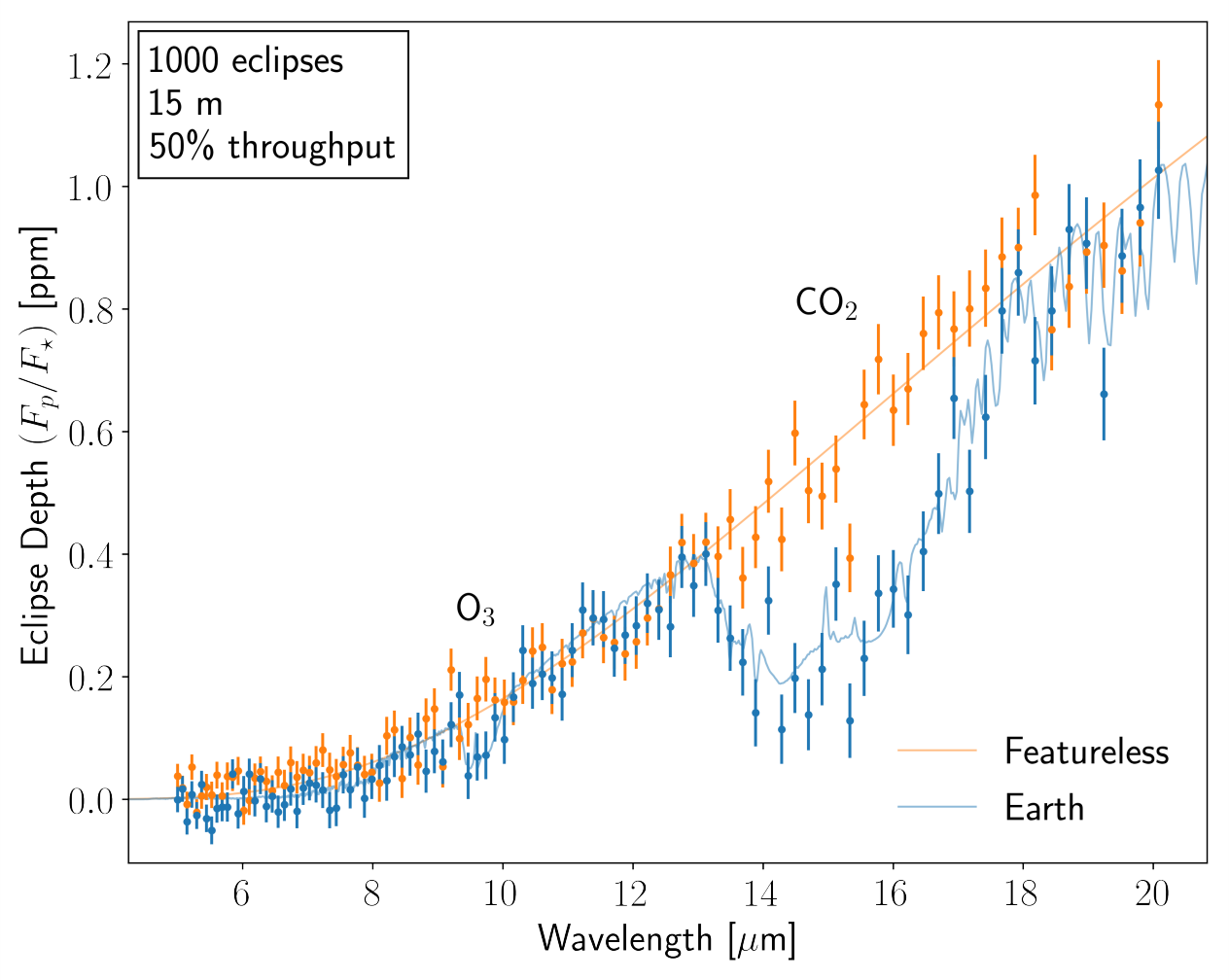
[24]:
```
planet.Tplan = 288
```

Now we can create a new `EclipseNoise` object for our featureless spectrum and calculate the photon count rates.

[25]:
```
enf = cg.EclipseNoise(tdur = tdur,
                      telescope = telescope,
                      planet = planet,
                      star = star,
                      ntran = ntran,
                      nout = nout,
                      wantsnr = wantsnr)

enf.run_count_rates(lam)
```

Let's plot our new featureless spectrum and compare it with Earth's emission spectrum.

[26]:
```
# Plot Spectrum
fig, ax = enf.plot_spectrum(SNR_threshold=0.0, Nsig=None,
                            err_kws={'fmt': '.', 'c': 'C1', 'alpha': 1},
                            plot_kws={'lw': 1.0, 'c': 'C1', 'alpha': 0.5, 'label' :
↪'Featureless'})

# Add Earth's emission spectrum to the axis
en.plot_spectrum(SNR_threshold=0.0, Nsig=None, ax0=ax,
                 err_kws={'fmt': '.', 'c': 'C0', 'alpha': 1},
                 plot_kws={'lw': 1.0, 'c': 'C0', 'alpha': 0.5, 'label' : 'Earth'})

# Add legend
leg = ax.legend(loc = 4)
leg.get_frame().set_alpha(0.0)

# Annotate molecules
ax.text(15.0, 0.8, r"CO$_2$", va = "center", ha = "center");
ax.text(9.6, 0.3, r"O$_3$", va = "center", ha = "center");
```



What about M-dwarfs?

So far we have looked at the extremely unrealistic scenario of observing Earth in secondary eclipse around the Sun. Now let's shift our focus to the late M-dwarf TRAPPIST-1 to see what the Earth would look like in secondary eclipse around this small, cool, dim star.

We'll start by modifying the effective temperature `Teff` and the radius `Rs` of the `star`:

[27]:
```
star.Teff = 2510
star.Rs = 0.117
```

Let's instantiate an `EclipseNoise` object but now we'll use the transit duration of TRAPPIST-1e and only observe 25 secondary eclipses.

[28]:
```
enm = cg.EclipseNoise(tdur = 3432.0,
                      telescope = telescope,
                      planet = planet,
                      star = star,
                      ntran = 25.0,
                      nout = nout,
                      wantsnr = wantsnr)
```

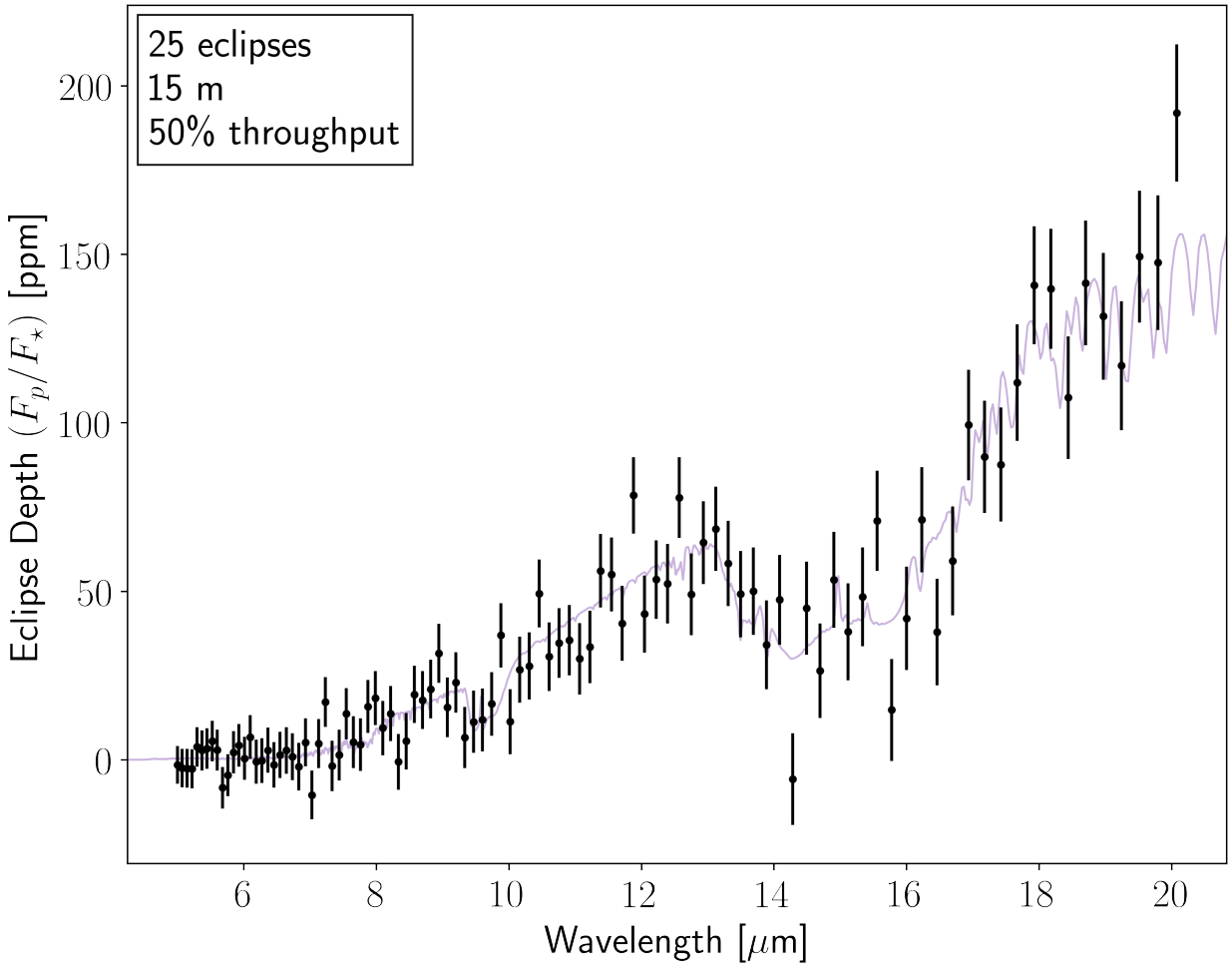
(continues on next page)

(continued from previous page)

```
enm.run_count_rates(lam, fplan)
```

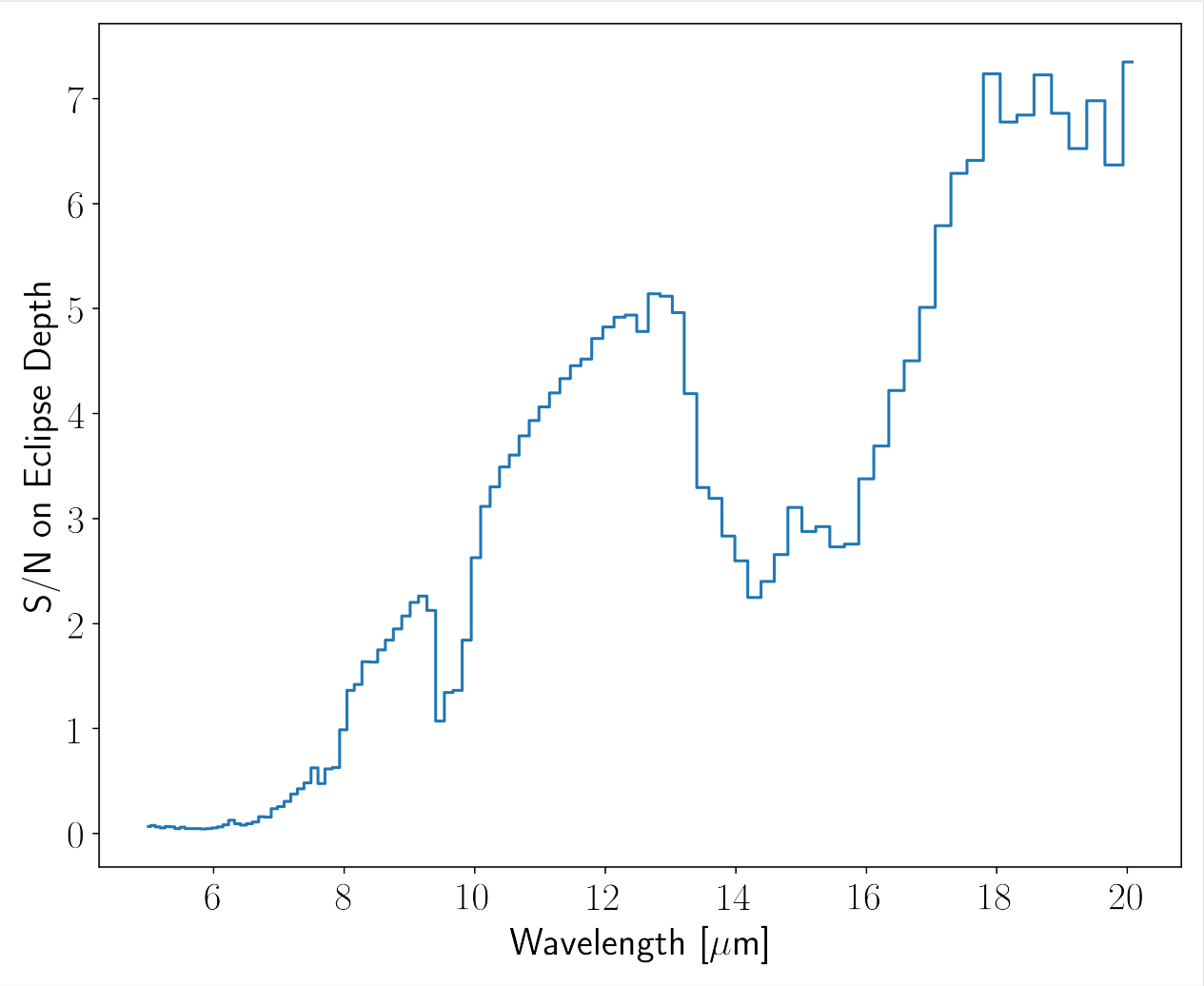
Here's the emission spectrum:

[29]:
```
fig, ax = enm.plot_spectrum(SNR_threshold=0.0, Nsig=None)
```

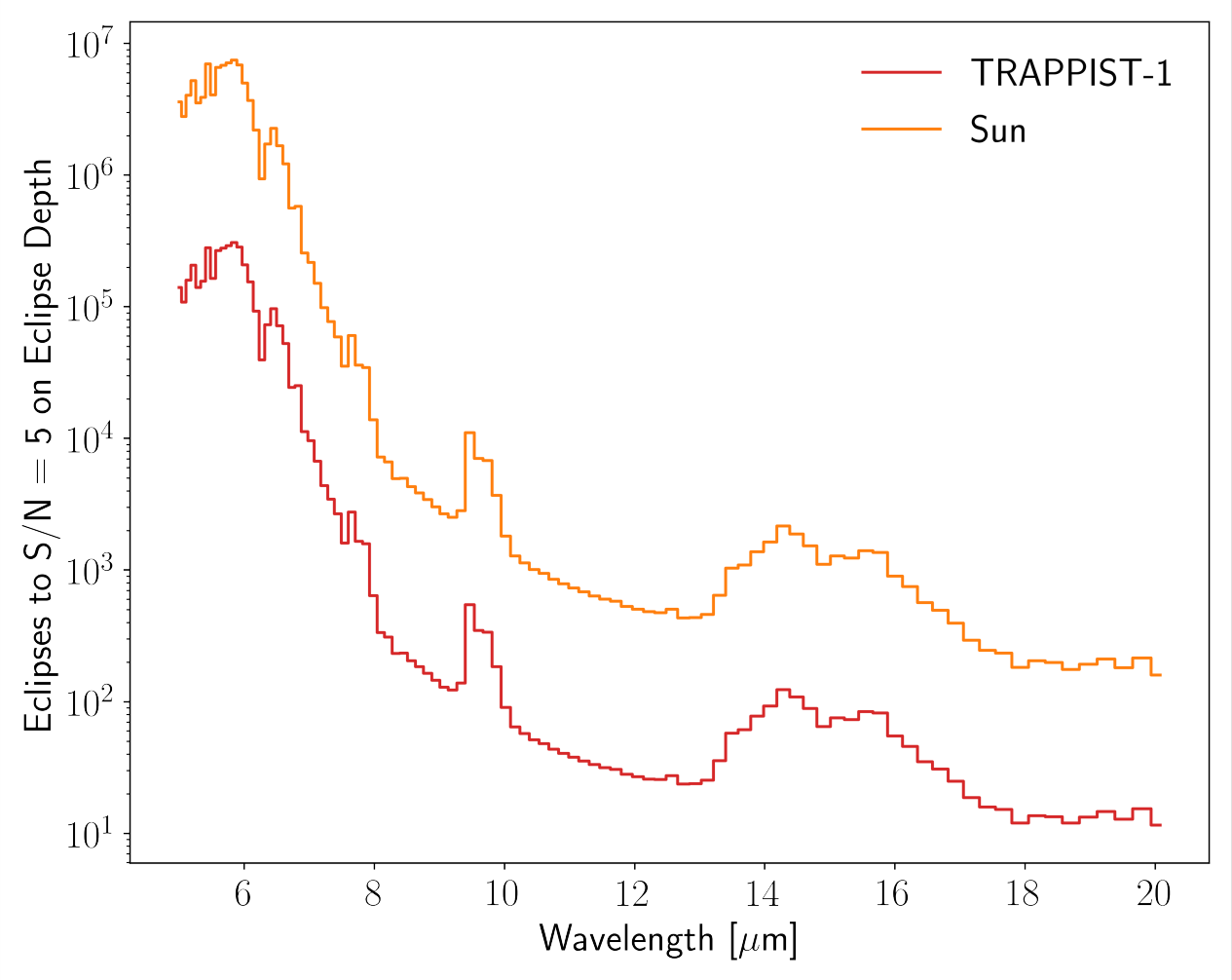


Now we're in business!

[30]:
```
fig, ax = enm.plot_SNRn()
```



[37]:
```
enm.recalc_wantsnr(wantsnr = 5.0)
en.recalc_wantsnr(wantsnr = 5.0)
fig, ax = enm.plot_ntran_to_wantsnr(plot_kws={'ls': 'steps-mid', 'alpha': 1.0, 'label
↪' : "TRAPPIST-1", 'c' : 'C3'})
en.plot_ntran_to_wantsnr(ax0 = ax, plot_kws={'ls': 'steps-mid', 'alpha': 1.0, 'label'␣
↪: "Sun", 'c' : 'C1'})
ax.legend(framealpha = 0.0);
```

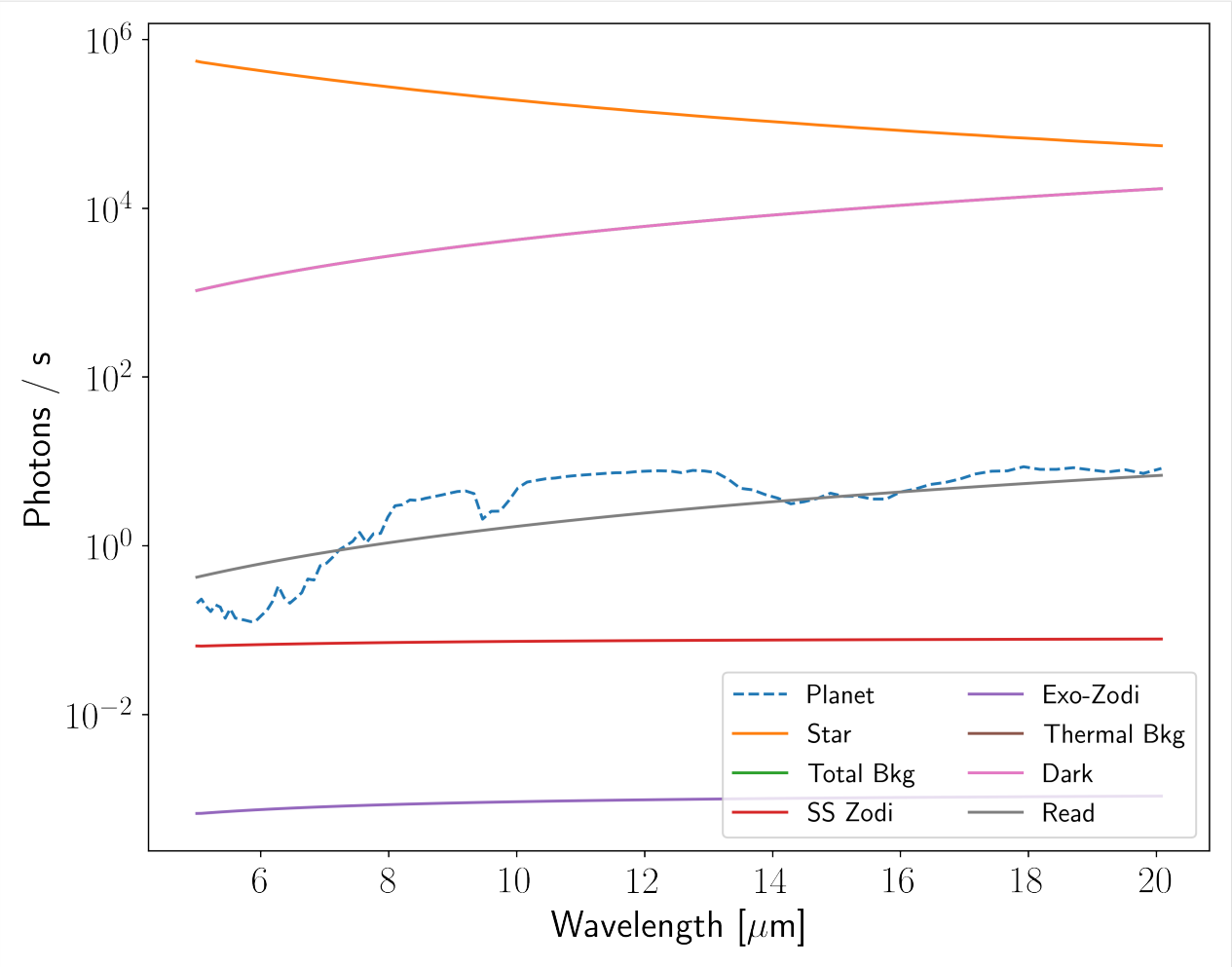


Compared to the Sun, studying Earth temperature planets in secondary eclipse around TRAPPIST-1 would require better than $10\times$ fewer transits to reach the same S/N on the spectrum.

Note that a S/N of 5 on each spectral element says nothing of the ability to detect molecules. The 15 μm CO_2 band as seen in the spectrum above is detected at greater than S/N=5 even though the S/N<5 *in each element* within the absorption band.

Finally, let's look at the photon count rates.

[38]:
```
fig, ax = enm.plot_count_rates()
```



Now the contrast is $\sim 100\times$ more favorable than the Earth-Sun system. These systems are by no means easy to study, but they are something that we can work with!

1.4 Scripts

A collection of scripts that demonstrate how to use `coronagraph`.

1.4.1 luvoir_demo.py

A demo simulation of Earth at 10 pc using a LUVOIR-like telescope setup.

`scripts.luvoir_demo.run()`
Run an example `coronagraph` spectrum assuming a space-based observatory with LUVOIR-like specifications.

Example

```
>>> import luvoir_demo
>>> luvoir_demo.run()
```

1.4.2 ground_demo.py

A demo simulation of Earth at 10 pc using a 30-m ground-based telescope setup.

scripts.ground_demo.**run**()
Run an example coronagraph spectrum assuming a ground-based observatory.

Example

```
>>> import ground_demo
>>> ground_demo.run()
```

1.4.3 transit_demo.py

A demo for modeling transmission spectra with a blank mask coronagraph (i.e. the coronagraph is not blocking the star's light).

scripts.transit_demo.**earth_analog_transits**(*d=10.0*, *ntran=10*, *nout=2*)
Simulate the transmission spectrum of Earth transiting a Sun-like star that is *d* parsecs away.

Parameters

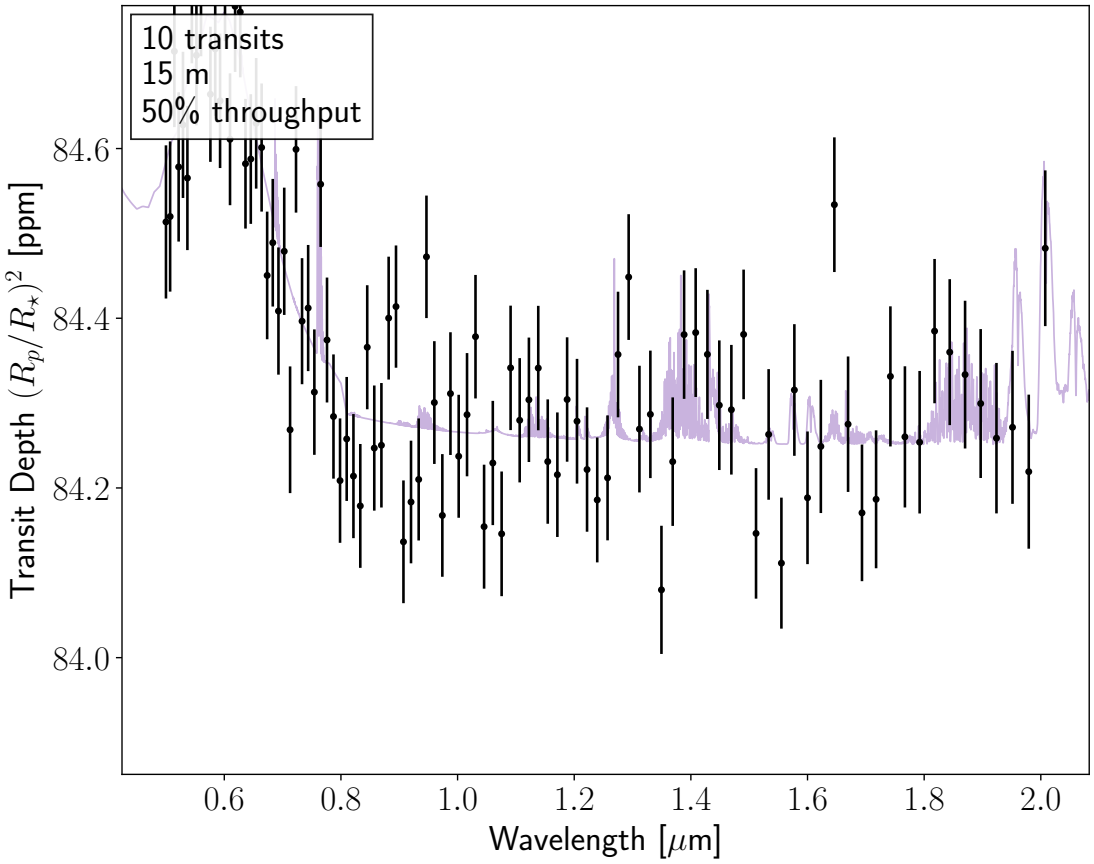
- **d** (*float*) – Distance to system [pc]
- **ntran** (*int*) – Number of transits
- **nout** (*int*) – Number of out-of-transit transit durations to observe

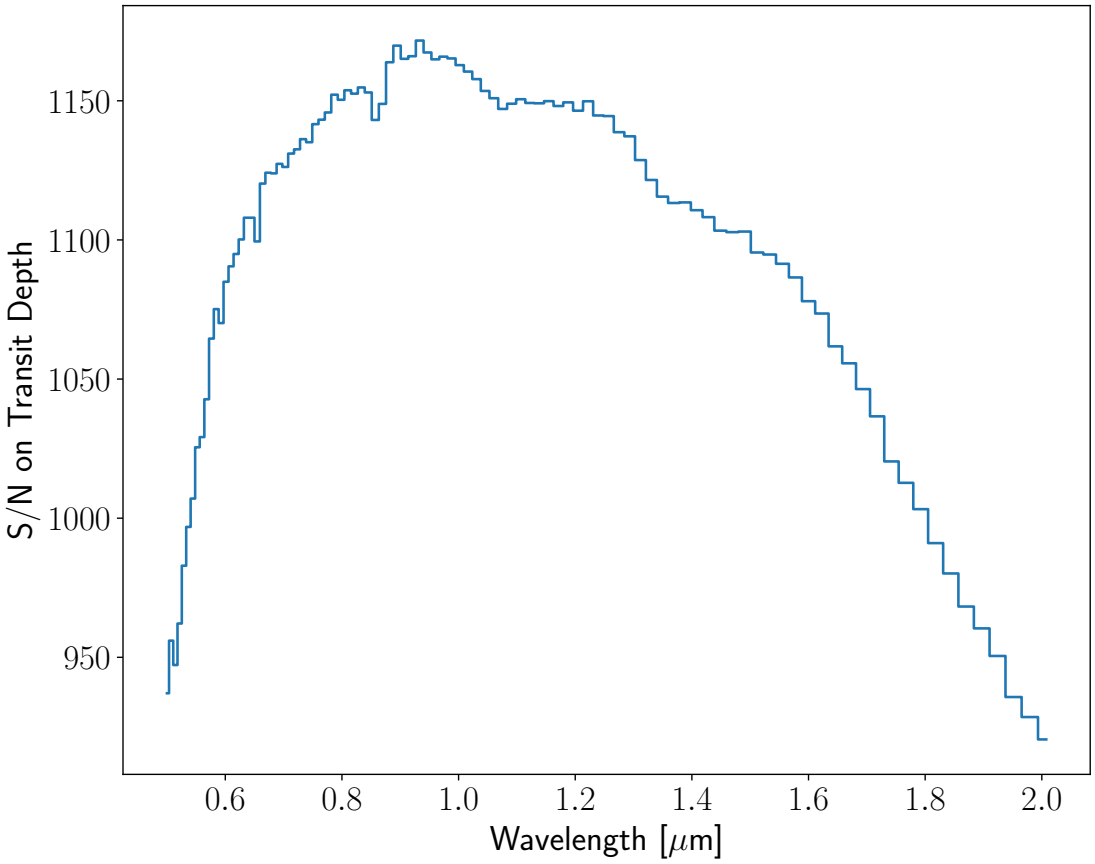
Example

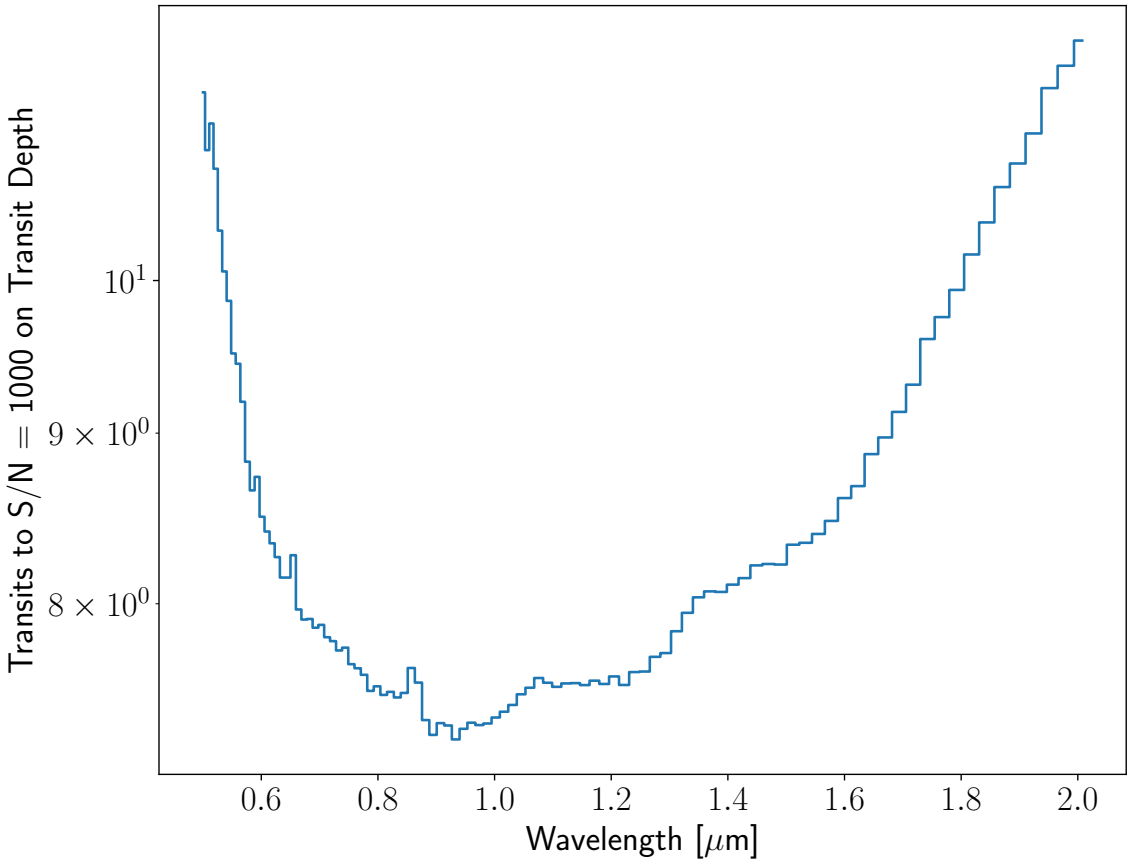
```
>>> from transit_demo import earth_analog_transits
>>> earth_analog_transits()
```

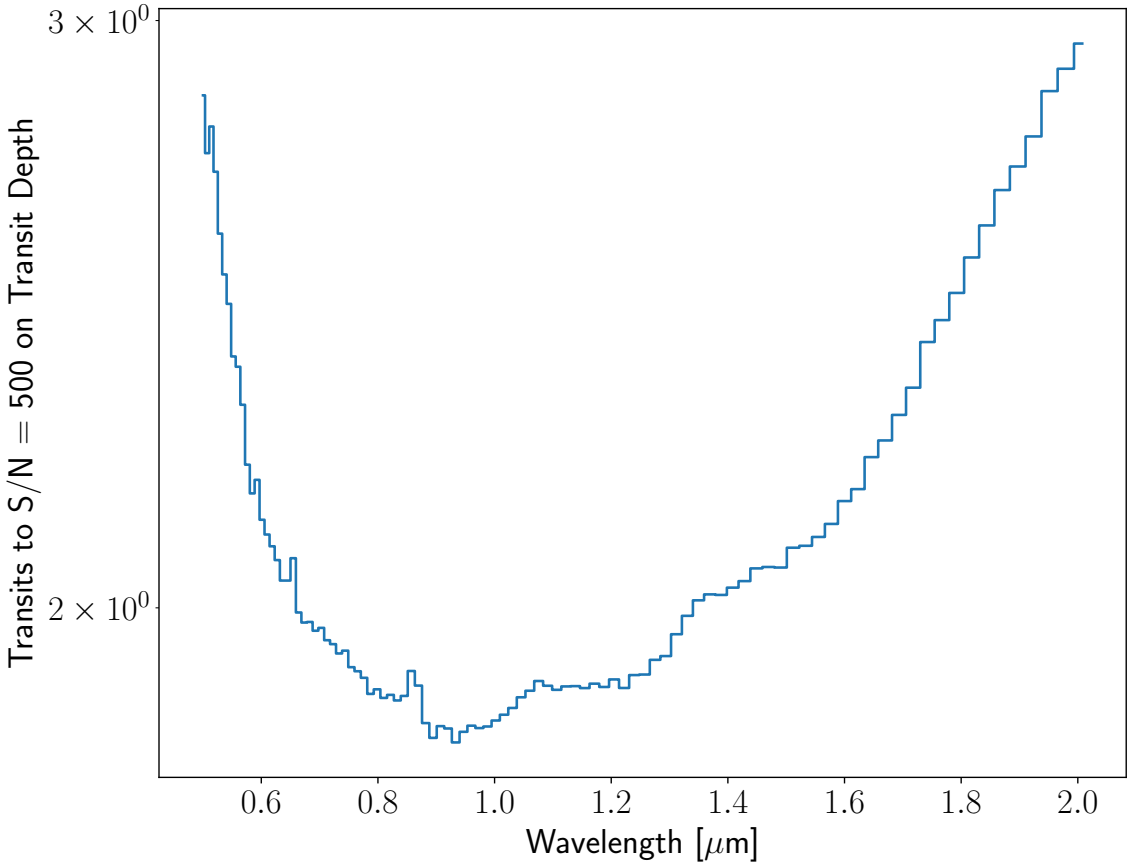
1.5 API

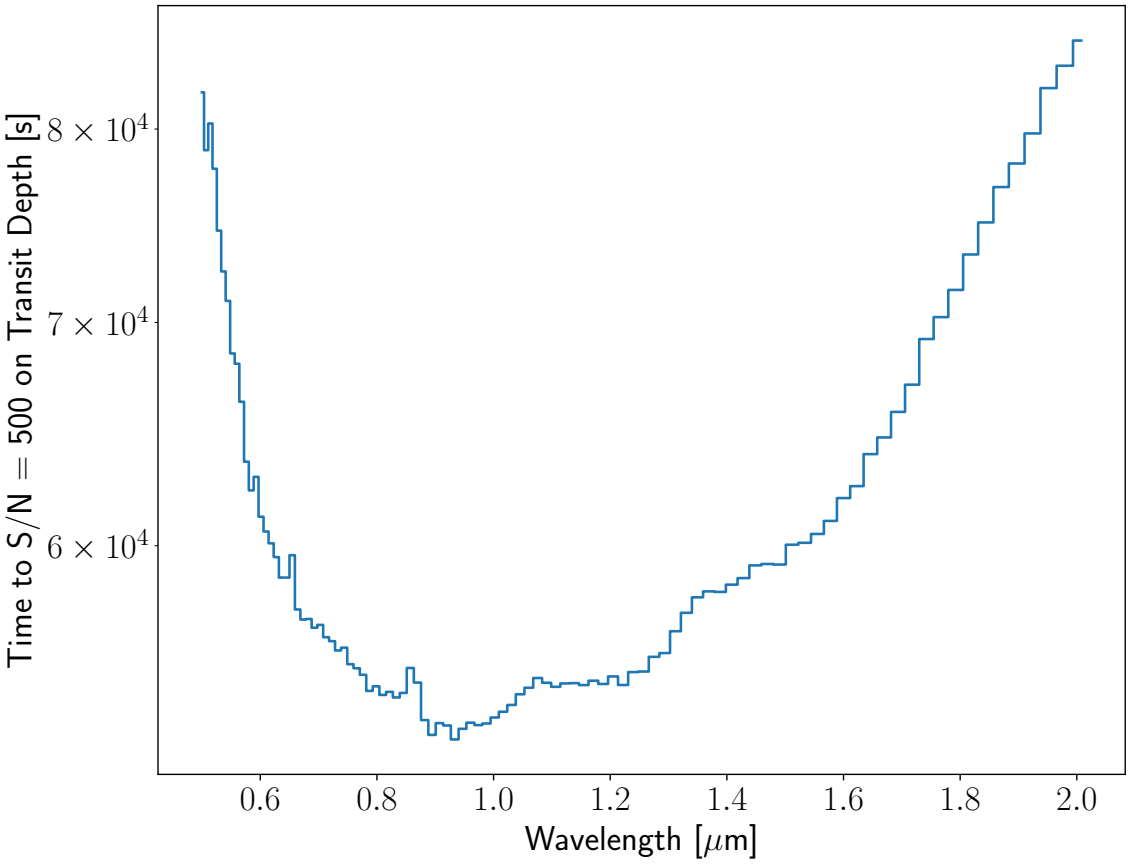
Detailed documentation of the Python code.

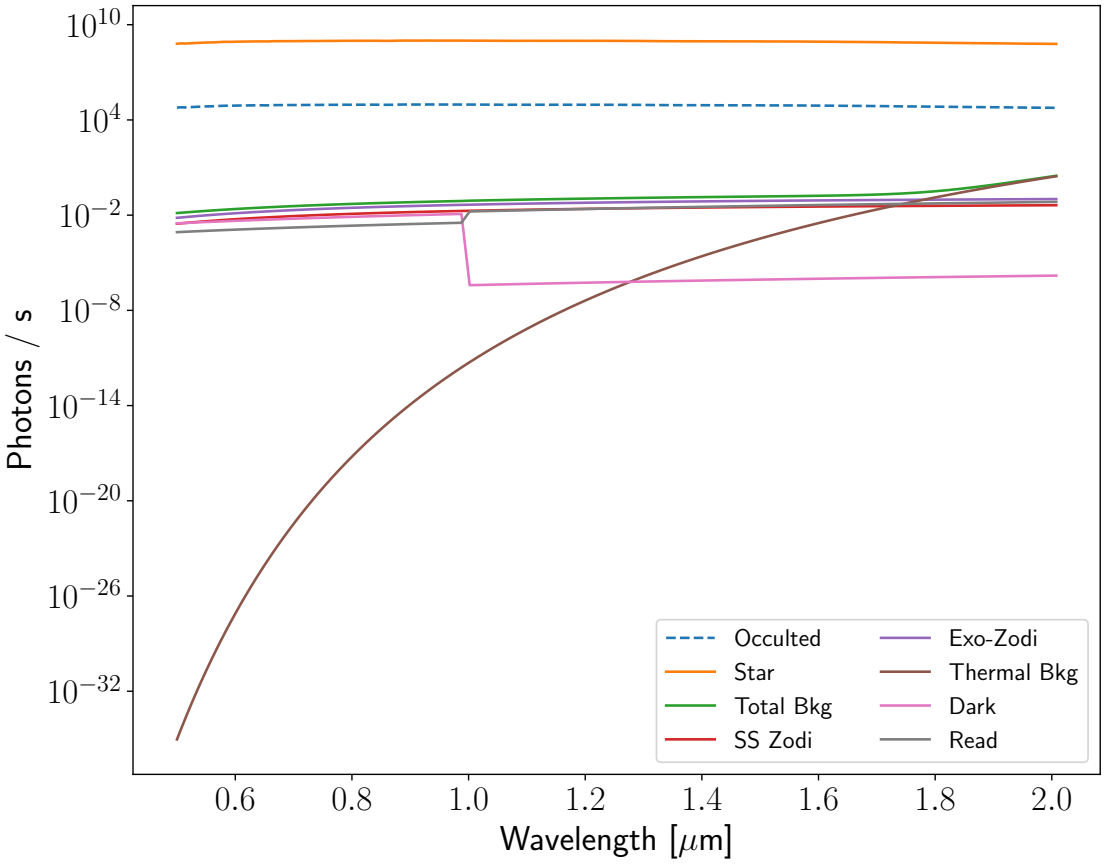

10 transits
15 m
50% throughput
84.6
84.4
84.2
84.0
Transit Depth $(R_p/R_\star)^2$ [ppm]
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]


S/N on Transit Depth
1150
1100
1050
1000
950
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]


Transits to S/N = 1000 on Transit Depth
10^1
9×10^0
8×10^0
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]


3×10^0
2×10^0
Transits to S/N = 500 on Transit Depth
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]


Time to S/N = 500 on Transit Depth [s]
8×10^4
7×10^4
6×10^4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]


10^{10}
10^{4}
10^{-2}
10^{-8}
10^{-14}
10^{-20}
10^{-26}
10^{-32}
Photons / s
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Wavelength [μm]
Occulted
Star
Total Bkg
SS Zodi
Exo-Zodi
Thermal Bkg
Dark
Read

1.5.1 Coronagraph Noise Modeling

The crux of `coronagraph` noise modeling is to determine the photon count rate incident upon the detector due to both the target planet and an assortment of different telescope, instrumental, and astrophysical noise sources. The following classes and functions serve as your interface to the photon count rate calculations. The core function for these calculations is *`count_rates()`*, but it may be accessed using the *`CoronagraphNoise`* object.

class coronagraph.count_rates.**CoronagraphNoise**(*telescope=<coronagraph.teleplanstar.Telescope object>*, *planet=<coronagraph.teleplanstar.Planet object>*, *star=<coronagraph.teleplanstar.Star object>*, *texp=10.0*, *wantsnr=10.0*, *FIX_OWA=False*, *COMPUTE_LAM=False*, *SILENT=False*, *NIR=False*, *THERMAL=True*, *GROUND=False*, *vod=False*, *set_fpa=None*, *roll_maneuver=True*)

The primary interface for `coronagraph` noise modeling. This object wraps around the functionality of *`count_rates()`*. Simply instantiate a *CoronagraphNoise* object by passing it `telescope`, `planet`, and `star` objects, and then call *`CoronagraphNoise.run_count_rates()`* to perform the photon count rate calculation.

Parameters

- **telescope** (`Telescope`) – Initialized object containing `Telescope` parameters
- **planet** (`Planet`) – Initialized object containing `Planet` parameters
- **star** (`Star`) – Initialized object containing `Star` parameters
- **texp** (*`float`*) – Exposure time for which to generate synthetic data [hours]
- **wantsnr** (*`float, optional`*) – Desired signal-to-noise ratio in each pixel
- **FIX_OWA** (*`bool, optional`*) – Set to fix OWA at `OWA*lammin/D`, as would occur if lenslet array is limiting the OWA
- **COMPUTE_LAM** (*`bool, optional`*) – Set to compute lo-res wavelength grid, otherwise the grid input as variable `lam` is used
- **SILENT** (*`bool, optional`*) – Set to suppress print statements
- **NIR** (*`bool, optional`*) – Re-adjusts pixel size in NIR, as would occur if a second instrument was designed to handle the NIR
- **THERMAL** (*`bool, optional`*) – Set to compute thermal photon counts due to telescope temperature
- **GROUND** (*`bool, optional`*) – Set to simulate ground-based observations through atmosphere
- **vod** (*`bool, optional`*) – "Valley of Death" red QE parameterization from Robinson et al. (2016)
- **set_fpa** (*`float, optional`*) – Specify the fraction of planetary signal in Airy pattern, default will calculate it from the photometric aperture size *X*
- **roll_maneuver** (*`bool, optional`*) – This assumes an extra factor of 2 hit to the background noise due to a telescope roll maneuver needed to subtract out the background. See Brown (2005) for more details.

Note: The results of the coronagraph noise calculation will become available as attributes of the *CoronagraphNoise* object after *CoronagraphNoise.run_count_rates()* is called.

run_count_rates (*Ahr*, *lamhr*, *solhr*)

Calculate the photon count rates and signal to noise on a coronagraph observation given a wavelength-dependent planetary geometric albedo and stellar flux density.

Parameters

- **Ahr** (*array*) – High-res, wavelength-dependent planetary geometric albedo
- **lamhr** (*array*) – High-res wavelength grid [um]
- **solhr** (*array*) – High-res TOA solar spectrum [W/m**2/um]

Calling `run_count_rates()` creates the following attributes for the `CoronagraphNoise` instance:

Variables

- **Ahr** (*array*) – High-res, wavelength-dependent planetary geometric albedo
- **lamhr** (*array*) – High-res wavelength grid [um]
- **solhr** (*array*) – High-res TOA solar spectrum [W/m**2/um]
- **lam** (*array*) – Observed wavelength grid [mum]
- **dlam** (*array*) – Observed wavelength grid widths [mum]
- **A** (*array*) – Planetary geometric albedo at observed resolution
- **Cratio** (*array*) – Planet-to-star flux contrast ratio
- **cp** (*array*) – Planetary photon count rate [photons/s]
- **csp** (*array*) – Speckle count rate [photons/s]
- **cz** (*array*) – Zodi photon count rate [photons/s]
- **cez** (*array*) – Exo-zodi photon count rate [photons/s]
- **cth** (*array*) – Thermal photon count rate [photons/s]
- **cD** (*array*) – Dark current photon count rate [photons/s]
- **cR** (*array*) – Read noise photon count rate [photons/s]
- **cc** (*array*) – Clock induced charge photon count rate [photons/s]
- **cb** (*array*) – Total background photon noise count rate [photons/s]
- **DtSNR** (*array*) – Integration time to `wantsnr` [hours]
- **SNRt** (*array*) – S/N in a `texp` hour exposure
- **Aobs** (*array*) – Observed albedo with noise
- **Asig** (*array*) – Observed uncertainties on albedo
- **Cobs** (*array*) – Observed Fp/Fs with noise
- **Csig** (*array*) – Observed uncertainties on Fp/Fs

make_fake_data (*texp=None*)

Make a fake/synthetic dataset by sampling from a Gaussian.

Parameters **texp** (*float, optional*) – Exposure time [hours]. If not provided, the `CoronagraphNoise.texp` will be used by default.

Calling `make_fake_data()` creates the following attributes for the `CoronagraphNoise` instance:

Variables

- **SNRt** (*array*) – S/N in a `texp` hour exposure
- **Aobs** (*array*) – Observed albedo with noise
- **Asig** (*array*) – Observed uncertainties on albedo
- **Cobs** (*array*) – Observed Fp/Fs with noise
- **Csig** (*array*) – Observed uncertainties on Fp/Fs

plot_spectrum(*SNR_threshold=1.0*, *Nsig=6.0*, *ax0=None*, *err_kws={'alpha': 1*, *'c': 'k'*, *'fmt': '.'}*, *plot_kws={'alpha': 0.5*, *'c': 'C4'*, *'lw': 1.0}*, *draw_box=True*)

Plot noised direct-imaging spectrum.

Parameters

- **SNR_threshold** (*float*) – Threshold SNR below which do not plot
- **Nsig** (*float*) – Number of standard deviations about median observed points to set yaxis limits
- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **err_kws** (*dic*) – Keyword arguments for *errorbar*
- **plot_kws** (*dic*) – Keyword arguments for *plot*
- **draw_box** (*bool*) – Draw important quantities in a box?

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

---

plot_SNR(*ax0=None*, *plot_kws={'ls': 'steps-mid'}*)

Plot the S/N on the planet as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

---

plot_time_to_wantsnr(*ax0=None*, *plot_kws={'alpha': 1.0*, *'ls': 'steps-mid'}*)

Plot the exposure time to get a SNR on the planet spectrum.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

---

coronagraph.count_rates.**count_rates**(*Ahr*, *lamhr*, *solhr*, *alpha*, *Phi*, *Rp*, *Teff*, *Rs*, *r*, *d*, *Nez*, *mode='IFS'*, *filter_wheel=None*, *lammin=0.4*, *lammax=2.5*, *Res=70.0*, *diam=10.0*, *Tput=0.2*, *C=1e-10*, *IWA=3.0*, *OWA=20.0*, *Tsys=260.0*, *Tdet=50.0*, *emis=0.9*, *De=0.0001*, *DNHpix=3.0*, *Re=0.1*, *Rc=0.0*, *Dtmax=1.0*, *X=1.5*, *qe=0.9*, *MzV=23.0*, *MezV=22.0*, *A_collect=None*, *diam_circumscribed=None*, *diam_inscribed=None*, *lam=None*, *dlam=None*, *Tput_lam=None*, *qe_lam=None*, *lammin_lenslet=None*, *wantsnr=10.0*, *FIX_OWA=False*, *COMPUTE_LAM=False*, *SILENT=False*, *NIR=False*, *THERMAL=False*, *GROUND=False*, *vod=False*, *set_fpa=None*, *CIRC=True*, *roll_maneuver=True*)

Runs coronagraph model (Robinson et al., 2016) to calculate planet and noise photon count rates for specified telescope and system parameters.

Parameters

- **Ahr** (*array*) – High-res, wavelength-dependent planetary geometric albedo
- **lamhr** (*array*) – High-res wavelength grid [um]
- **solhr** (*array*) – High-res TOA solar spectrum [W/m**2/um]
- **alpha** (*float*) – Planet phase angle [deg]
- **Phi** (*float*) – Planet phase function
- **Rp** (*float*) – Planet radius [R_earth]
- **Teff** (*float*) – Stellar effective temperature [K]
- **Rs** (*float*) – Stellar radius [R_sun]
- **r** (*float*) – Planet semi-major axis [AU]
- **d** (*float*) – Distance to observed star-planet system [pc]
- **Nez** (*float*) – Number of exozodis in exoplanetary disk
- **mode** (*str, optional*) – Telescope observing mode: “IFS” or “Imaging”
- **filter_wheel** (*Wheel, optional*) – Wheel object containing imaging filters
- **lammin** (*float, optional*) – Minimum wavelength [um]
- **lammax** (*float, optional*) – Maximum wavelength [um]
- **Res** (*float, optional*) – Instrument spectral resolution (`lam / dlam`)
- **diam** (*float, optional*) – Telescope diameter [m]
- **Tput** (*float, optional*) – Telescope and instrument throughput

- **C** (*float, optional*) – Coronagraph design contrast
- **IWA** (*float, optional*) – Coronagraph Inner Working Angle (`lam / diam`)
- **OWA** (*float, optional*) – Coronagraph Outer Working Angle (`lam / diam`)
- **Tsys** (*float, optional*) – Telescope mirror temperature [K]
- **Tdet** (*float, optional*) – Telescope detector temperature [K]
- **emis** (*float, optional*) – Effective emissivity for the observing system (of order unity)
- **De** (*float, optional*) – Dark current [counts/s]
- **DNHpix** (*float, optional*) – Number of horizontal/spatial pixels for dispersed spectrum
- **Re** (*float, optional*) – Read noise counts per pixel
- **Rc** (*float, optional*) – Clock induced charge [counts/pixel/photon]
- **Dtmax** (*float, optional*) – Detector maximum exposure time [hours]
- **X** (*float, optional*) – Width of photometric aperture (`lam / diam`)
- **qe** (*float, optional*) – Detector quantum efficiency
- **MzV** (*float, optional*) – V-band zodiacal light surface brightness [mag/arcsec**2]
- **MezV** (*float, optional*) – V-band exozodiacal light surface brightness [mag/arcsec**2]
- **A_collect** (*float, optional*) – Mirror collecting area (m**2) (uses $\pi(D/2)^2$ by default)
- **diam_circumscribed** (*float, optional*) – Circumscribed telescope diameter [m] used for IWA and OWA (uses *diam* if *None* provided)
- **diam_inscribed** (*float, optional*) – Inscribed telescope diameter [m] used for lenslet calculations (uses *diam* if *None* provided)
- **lam** (*array-like, optional*) – Wavelength grid for spectrograph [microns] (uses `lammin`, `lammax`, and `resolution` to determine if `None` provided)
- **dlam** (*array-like, optional*) – Wavelength grid *widths* for spectrograph [microns] (uses `lammin`, `lammax`, and `resolution` to determine if `None` provided)
- **Tput_lam** (*tuple of arrays*) – Wavelength-dependent throughput e.g. `(wls, tputs)`
- **qe_lam** (*tuple of arrays*) – Wavelength-dependent throughput e.g. `(wls, qe)`
- **lammin_lenslet** (*float, optional*) – Minimum wavelength to use for lenslet calculation (default is `lammin`)
- **wantsnr** (*float, optional*) – Desired signal-to-noise ratio in each pixel
- **FIX_OWA** (*bool, optional*) – Set to fix OWA at `OWA*lammin/D`, as would occur if lenslet array is limiting the OWA
- **COMPUTE_LAM** (*bool, optional*) – Set to compute lo-res wavelength grid, otherwise the grid input as variable `lam` is used
- **SILENT** (*bool, optional*) – Set to suppress print statements

- **NIR** (*bool, optional*) – Re-adjusts pixel size in NIR, as would occur if a second instrument was designed to handle the NIR
- **THERMAL** (*bool, optional*) – Set to compute thermal photon counts due to telescope temperature
- **GROUND** (*bool, optional*) – Set to simulate ground-based observations through atmosphere
- **vod** (*bool, optional*) – "Valley of Death" red QE parameterization from Robinson et al. (2016)
- **set_fpa** (*float, optional*) – Specify the fraction of planetary signal in Airy pattern, default will calculate it from the photometric aperture size *X*
- **CIRC** (*bool, optional*) – Set to use a circular aperture
- **roll_maneuver** (*bool, optional*) – This assumes an extra factor of 2 hit to the background noise due to a telescope roll maneuver needed to subtract out the background. See Brown (2005) for more details.

Returns

- **lam** (*ndarray*) – Observational wavelength grid [um]
- **dlam** (*ndarray*) – Observational spectral element width [um]
- **A** (*ndarray*) – Planetary geometric albedo on lam grid
- **q** (*ndarray*) – Quantum efficiency grid
- **Cratio** (*ndarray*) – Planet-star contrast ratio
- **cp** (*ndarray*) – Planetary photon count rate on detector [1/s]
- **csp** (*ndarray*) – Speckle photon count rate on detector [1/s]
- **cz** (*ndarray*) – Zodiacal photon count rate on detector [1/s]
- **cez** (*ndarray*) – Exozodiacal photon count rate on detector [1/s]
- **cD** (*ndarray*) – Dark current photon count rate on detector [1/s]
- **cR** (*ndarray*) – Read noise photon count rate on detector [1/s]
- **cth** (*ndarray*) – Instrument thermal photon count rate on detector [1/s]
- **cc** (*ndarray*) – Clock induced charge photon count rate [1/s]
- **DtSNR** (*ndarray*) – Exposure time required to get desired S/N (wantsnr) [hours]

1.5.2 Observational Routines

The following functions provide additional mock observing features for use with the coronagraph model.

coronagraph.observe.**get_earth_reflect_spectrum**()

Get the geometric albedo spectrum of the Earth around the Sun. This was produced by Tyler Robinson using the VPL Earth Model (Robinson et al., 2011)

Returns

- **lamhr** (*numpy.ndarray*)
- **Ahr** (*numpy.ndarray*)
- **fstar** (*numpy.ndarray*)

coronagraph.observe.**planetzoo_observation**(*name='earth'*, *telescope=<coronagraph.teleplanstar.Telescope object>*, *planet=<coronagraph.teleplanstar.Planet object>*, *itime=10.0*, *planetdir='/home/docs/checkouts/readthedocs.org/user_builds/coronagr packages/coronagraph-1.0-py3.5.egg/coronagraph/planets/'*, *plot=False*, *savedata=False*, *saveplot=False*, *ref_lam=0.55*, *THERMAL=False*)

Observe the Solar System planets as if they were exoplanets.

Parameters

- **name** (*str (optional)*) –

  Name of the planet. Possibilities include: "venus", "earth", "archean", "mars", "early-mars", "hazyarchean", "earlyvenus", "jupiter", "saturn", "uranus", "neptune"

- **telescope** (Telescope *(optional)*) – Telescope object to be used for observation
- **planet** (Planet *(optional)*) – Planet object to be used for observation
- **itime** (*float (optional)*) – Integration time (hours)
- **planetdir** (*str*) – Location of planets/ directory
- **plot** (*bool (optional)*) – Make plot flag
- **savedata** (*bool (optional)*) – Save output as data file
- **saveplot** (*bool (optional)*) – Save plot as PDF
- **ref_lam** (*float (optional)*) – Wavelength at which SNR is computed

Returns

- **lam** (*array*) – Observed wavelength array (microns)
- **spec** (*array*) – Observed reflectivity spectrum
- **sig** (*array*) – Observed 1-sigma error bars on spectrum

Example

```
>>> from coronagraph.observe import planetzoo_observation
>>> lam, spec, sig = planetzoo_observation(name = "venus", plot = True)
```

```
>>> lam, spec, sig = planetzoo_observation(name = "earth", plot = True)
```

```
>>> lam, spec, sig = planetzoo_observation(name = "mars", plot = True)
```

coronagraph.observe.**generate_observation**(*wlhr*, *Ahr*, *solhr*, *itime*, *telescope*, *planet*, *star*, *ref_lam=0.55*, *tag=''*, *plot=True*, *saveplot=False*, *savedata=False*, *THERMAL=False*, *wantsnr=10*)

(Depreciated) Generic wrapper function for *count_rates*.

Parameters

- **wlhr** (*float*) – Wavelength array (microns)
- **Ahr** (*float*) – Geometric albedo spectrum array

- **itime** (*float*) – Integration time (hours)
- **telescope** (Telescope) – Telescope object
- **planet** (Planet) – Planet object
- **star** (Star) – Star object
- **tag** (*string*) – ID for output files
- **plot** (*boolean*) – Set to True to make plot
- **saveplot** (*boolean*) – Set to True to save the plot as a PDF
- **savedata** (*boolean*) – Set to True to save data file of observation

Returns

- **lam** (*array*) – Wavelength grid for observed spectrum
- **dlam** (*array*) – Wavelength grid widths for observed spectrum
- **A** (*array*) – Low res albedo spectrum
- **spec** (*array*) – Observed albedo spectrum
- **sig** (*array*) – One sigma errorbars on albedo spectrum
- **SNR** (*array*) – SNR in each spectral element

Note: If *saveplot=True* then plot will be saved If *savedata=True* then data will be saved

---

coronagraph.observe.**plot_coronagraph_spectrum**(*wl*, *ofrat*, *sig*, *itime*, *d*, *ref_lam*, *SNR*, *truth=None*, *xlim=None*, *ylim=None*, *title=''*, *save=False*, *tag=''*)

Plot synthetic data from the coronagraph model

Parameters

- **wl** (*array-like*) – Wavelength grid [microns]
- **ofrat** (*array-like*) – Observed contrast ratio (with noise applied)
- **sig** (*array-like*) – One-sigma errors on ofrat
- **itime** (*float*) – Integration time for calculated observation [hours]
- **d** (*float*) – Distance to system [pc]
- **ref_lam** (*float*) – Reference wavelength [microns]
- **SNR** (*array*) – Signal-to-noise
- **truth** (*array-like (optional)*) – True contrast ratio
- **xlim** (*list (optional)*) – Plot x-axis limits
- **ylim** (*list (optional)*) – Plot y-axis limits
- **title** (*str (optional)*) – Plot title and saved plot name
- **save** (*bool (optional)*) – Set to save plot
- **tag** (*str (optional)*) – String to append to saved file

Returns

- **fig** (*matplotlib.Figure*)

- **ax** (*matplotlib.Axis*)

Note: Only returns `fig, ax` if `save = False`

`coronagraph.observe.`**`process_noise`**(*Dt*, *Cratio*, *cp*, *cb*)
Computes SNR, noised data, and error on noised data.

Parameters

- **`Dt`** (*`float`*) – Telescope integration time in seconds
- **`Cratio`** (*`array`*) – Planet/Star flux ratio in each spectral bin
- **`cp`** (*`array`*) – Planet Photon count rate in each spectral bin
- **`cb`** (*`array`*) – Background Photon count rate in each spectral bin

Returns

- **cont** (*array*) – Noised Planet/Star flux ratio in each spectral bin
- **sigma** (*array*) – One-sigma errors on flux ratio in each spectral bin
- **SNR** (*array*) – Signal-to-noise ratio in each spectral bin

`coronagraph.observe.`**`random_draw`**(*val*, *sig*)
Draw fake data points from model `val` with errors `sig`

`coronagraph.observe.`**`interp_cont_over_band`**(*lam*, *cp*, *icont*, *iband*)
Interpolate the continuum of a spectrum over a masked absorption or emission band.

Parameters

- **`lam`** (*`array`*) – Wavelength grid (abscissa)
- **`cp`** (*`array`*) – Planet photon count rates or any spectrum
- **`icont`** (*`list`*) – Indicies of continuum (neighboring points)
- **`iband`** (*`list`*) – Indicies of spectral feature (the band)

Returns **ccont** – Continuum planet photon count rates across spectral feature, where len(ccont) == len(iband)

Return type list

`coronagraph.observe.`**`exptime_band`**(*cp*, *ccont*, *cb*, *iband*, *SNR=5.0*)
Calculate the exposure time necessary to get a given S/N on a molecular band following Eqn 7 from Robinson et al. (2016),

$$\mathrm{S/N_{band}} = \frac{\sum_j c_{\mathrm{cont},j} - c_{\mathrm{p},j}}{\sqrt{\sum_j c_{\mathrm{p},j} + 2c_{\mathrm{b},j}}}\,\Delta t_{\mathrm{exp}}^{1/2}$$

where the sum is over all spectral elements (denoted by subscript "j") within the molecular band.

Parameters

- **`cp`** – Planet count rate
- **`ccont`** – Continuum count rate
- **`cb`** – Background count rate
- **`iband`** – Indicies of molecular band

- **SNR** – Desired signal-to-noise ratio on molecular band

Returns **texp** – Telescope exposure time [hrs]

Return type float

coronagraph.observe.**plot_interactive_band**(*lam*, *Cratio*, *cp*, *cb*, *itime=None*, *SNR=5.0*)

Makes an interactive spectrum plot for the user to identify all observed spectral points that make up a molecular band. Once the plot is active, press 'c' then select neighboring points in the Continuum, press 'b' then select all points in the Band, then press 'd' to perform the calculation.

Parameters

- **lam** (*array*) – Wavelength grid
- **Cratio** (*array*) – Planet-to-star flux contrast ratio
- **cp** (*array*) – Planetary photon count rate
- **cb** (*array*) – Background photon count rate
- **itime** (*float (optional)*) – Fiducial exposure time for which to calculate the SNR
- **SNR** (*float (optional)*) – Fiducial SNR for which to calculate the exposure time

1.5.3 Telescopes, Planets, and Stars

The coronagraph model relies on numerous parameters describing the telescope, planet, and star used for each calculation. Below *Telescope*, *Planet*, and *Star* classes are listed, which can be instantiated and passed along to noise calculations.

class coronagraph.teleplanstar.**Telescope**(*mode='IFS'*, *lammin=0.3*, *lammax=2.0*, *R=70.0*, *Tput=0.2*, *D=8.0*, *Tsys=260.0*, *Tdet=50.0*, *IWA=0.5*, *OWA=30000.0*, *emis=0.9*, *C=1e-10*, *De=0.0001*, *DNHpix=3.0*, *Re=0.1*, *Rc=0.0*, *Dtmax=1.0*, *X=0.7*, *q=0.9*, *filter_wheel=None*, *aperture='circular'*, *A_collect=None*, *Tput_lam=None*, *qe_lam=None*, *lammin_lenslet=None*, *diam_circumscribed=None*, *diam_inscribed=None*, *lam=None*, *dlam=None*)

A class to represent a telescope object and all design specifications therein

Parameters

- **mode** (*str*) – Telescope observing modes: 'IFS', 'Imaging'
- **lammin** (*float*) – Minimum wavelength (um)
- **lammax** (*float*) – Maximum wavelength (um)
- **R** (*float*) – Spectral resolution (lambda / delta-lambda)
- **Tsys** (*float*) – Telescope temperature (K)
- **D** (*float*) – Telescope diameter (m)
- **emis** (*float*) – Telescope emissivity
- **IWA** (*float*) – Inner Working Angle (lambda/D)
- **OWA** (*float*) – Outer Working Angle (lambda/D)
- **Tput** (*float*) – Telescope throughput
- **C** (*float*) – Raw Contrast

- **De** (*float*) – Dark current (s**-1)
- **DNHpix** (*float*) – Horizontal pixel spread of IFS spectrum
- **Re** (*float*) – Read noise per pixel
- **Rc** (*float, optional*) – Clock induced charge [counts/pixel/photon]
- **Dtmax** (*float*) – Maximum exposure time (hr)
- **X** (*float*) – Size of photometric aperture (lambda/D)
- **q** (*float*) – Quantum efficiency
- **filter_wheel** (Wheel *(optional)*) – Wheel object containing imaging filters
- **aperture** (*str*) – Aperture type ("circular" or "square")
- **A_collect** (*float*) – Mirror collecting area (m**2) if different than $\pi(D/2)^2$
- **diam_circumscribed** (*float, optional*) – Circumscribed telescope diameter [m] used for IWA and OWA (uses *diam* if *None* provided)
- **diam_inscribed** (*float, optional*) – Inscribed telescope diameter [m] used for lenslet calculations (uses *diam* if *None* provided)
- **Tput_lam** (*tuple of arrays*) – Wavelength-dependent throughput e.g. `(wls, tputs)`. Note that if `Tput_lam` is used the end-to-end throughput will equal the convolution of `Tput_lam[1]` with `Tput`.
- **qe_lam** (*tuple of arrays*) – Wavelength-dependent throughput e.g. `(wls, qe)`. Note that if `qe_lam` is used the total quantum efficiency will equal the convolution of `qe_lam[1]` with `q`.
- **lammin_lenslet** (*float, optional*) – Minimum wavelength to use for lenslet calculation (default is `lammin`)
- **lam** (*array-like, optional*) – Wavelength grid for spectrograph [microns] (uses `lammin`, `lammax`, and `resolution` to determine if `None` provided)
- **dlam** (*array-like, optional*) – Wavelength grid *widths* for spectrograph [microns] (uses `lammin`, `lammax`, and `resolution` to determine if `None` provided)

default_luvoir()
Initialize telescope object using current LUVOIR parameters (Not decided!)

default_habex()
Initialize telescope object using current HabEx parameters (Not decided!)

default_wfirst()
Initialize telescope object using current WFIRST parameters (Not decided!)

classmethod default_luvoir()

classmethod default_habex()

classmethod default_wfirst()

property mode

property filter_wheel

class coronagraph.teleplanstar.**Planet**(*name='earth'*, *star='sun'*, *d=10.0*, *Nez=1.0*, *Rp=1.0*, *a=1.0*, *alpha=90.0*, *MzV=23.0*, *MezV=22.0*)
A class to represent a planet and all associated parameters of the planet to be observed.

Parameters

- **name** (*string*) – Planet name from database
- **star** (*string*) – Stellar type of planet host star
- **d** (*float*) – Distance to system (pc)
- **Nez** (*float*) – Number of exzodis (zodis)
- **Rp** (*float*) – Radius of planet (Earth Radii)
- **a** (*float*) – Semi-major axis (AU)
- **alpha** (*float*) – Phase angle (deg)
- **Phi** (*float*) – Lambertian phase function
- **MzV** (*float*) – Zodiacal light surface brightness (mag/arcsec**2)
- **MezV** (*float*) – exozodiacal light surface brightness (mag/arcsec**2)

from_file()
Initialize object using planet parameters in the Input file

property alpha

property Phi

class coronagraph.teleplanstar.**Star**(*Teff=5780.0*, *Rs=1.0*)
A class to represent the stellar host for an exoplanet observation

Parameters

- **Teff** (*float*) – Stellar effective temperature [K]
- **Rs** (*float*) – Stellar radius [Solar Radii]

1.5.4 Noise Routines

Routines for simulating *coronagraph* model noise terms are listed below. These functions are used within the coronagraph.CoronagraphNoise and *coronagraph.count_rates()* functions, but are provided here for independent use.

The most important functions are the individual photon count rate terms due to different photon sources. This including photons from the planet *cplan()*, from zodiacal light *czodi()* and exo-zodiacal light *cezodi()*, from coronagraph speckles *cspeck()*, from dark current *cdark()* and read noise *cread()*, from thermal emission from the telescope mirror *ctherm()*, and from clock-induced charge *ccic()*.

Optional ground-based telescope noise modeling includes extra terms for the emission from Earth's atmosphere incident on the telescope, *ctherm_earth()* (also see *get_sky_flux()*), and an additional throughput term due to atmospheric extinction *set_atmos_throughput()*.

Finally, there are some extra convenience functions: Calculate the fraction of Airy power contained in square or circular aperture using *f_airy()*; Construct a wavelength grid by specifying either a spectral resolving power or a fixed wavelength bandwidth using *construct_lam()*; Calculate the Lambertian Phase Function of a planet from the phase angle using *lambertPhaseFunction()*; Calculate the Planck blackbody radiance given temperature and wavelength using *planck()*.

coronagraph.noise_routines.**Fstar**(*lam*, *Teff*, *Rs*, *d*, *AU=False*)
Stellar flux function

Parameters

- **lam** (*float or array-like*) – Wavelength [um]
- **Teff** (*float*) – Stellar effective temperature [K]

- **Rs** – Stellar radius [solar radii]
- **d** – Distance to star [pc]
- **AU** (*bool, optional*) – Flag that indicates d is in AU

Returns **Fstar** – Stellar flux [W/m**2/um]

Return type float or array-like

coronagraph.noise_routines.**Fplan**(*A*, *Phi*, *Fstar*, *Rp*, *d*, *AU=False*)

Planetary flux function

Parameters

- **A** (*float or array-like*) – Planetary geometric albedo
- **Phi** (*float*) – Planetary phase function
- **Fstar** (*float or array-like*) – Stellar flux [W/m**2/um]
- **Rp** (*float*) – Planetary radius [Earth radii]
- **d** (*float*) – Distance to star [pc]
- **AU** (*bool, optional*) – Flag that indicates d is in AU

Returns **Fplan** – Planetary flux [W/m**2/um]

Return type float or array-like

coronagraph.noise_routines.**FpFs**(*A*, *Phi*, *Rp*, *r*)

Planet-star flux ratio (Equation 11 from Robinson et al. 2016).

$$\frac{F_{p,\lambda}}{F_{\mathrm{s},\lambda}} = A\Phi(\alpha)\left(\frac{R_\mathrm{p}}{r}\right)^2$$

Parameters

- **A** (*float or array-like*) – Planetary geometric albedo
- **Phi** (*float*) – Planetary phase function
- **Rp** (*float*) – Planetary radius [Earth radii]
- **r** (*float*) – Planetary orbital semi-major axis [AU]

Returns **FpFs** – Planet-star flux ratio

Return type float or array-like

coronagraph.noise_routines.**cstar**(*q*, *fpa*, *T*, *lam*, *dlam*, *Fstar*, *D*)

Stellar photon count rate (not used with coronagraph)

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **fpa** (*float*) – Fraction of planetary light that falls within photometric aperture
- **T** (*float*) – Telescope and system throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **Fplan** (*float or array-like*) – Planetary flux [W/m**2/um]
- **D** (*float*) – Telescope diameter [m]

Returns **cs** – Stellar photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**cplan**(*q*, *fpa*, *T*, *lam*, *dlam*, *Fplan*, *D*)

Exoplanetary photon count rate (Equation 12 from Robinson et al. 2016)

$$c_{\mathrm{p}} = \pi q f_{\mathrm{pa}} \mathcal{T} \frac{\lambda}{hc} F_{\mathrm{p},\lambda}(d) \Delta\lambda \left(\frac{D}{2}\right)^2$$

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **fpa** (*float*) – Fraction of planetary light that falls within photometric aperture
- **T** (*float*) – Telescope and system throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **Fplan** (*float or array-like*) – Planetary flux [W/m**2/um]
- **D** (*float*) – Telescope diameter [m]

Returns **cplan** – Exoplanetary photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**czodi**(*q*, *X*, *T*, *lam*, *dlam*, *D*, *Mzv*, *SUN=False*, *CIRC=False*)

Zodiacal light count rate (Equation 15 from Robinson et al. 2016)

$$c_{\mathrm{z}} = \pi q \mathcal{T} \Omega \Delta\lambda \frac{\lambda}{hc} \left(\frac{D}{2}\right)^2 \frac{F_{\odot,\lambda}(1\ \mathrm{AU})}{F_{\odot,V}(1\ \mathrm{AU})}\ F_{0,V} 10^{-M_{\mathrm{z},V}/2.5}$$

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **X** (*float*) – Size of photometric aperture (lambda/D)
- **T** (*float*) – Telescope and system throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **D** (*float*) – Telescope diameter [m]
- **MzV** (*float*) – Zodiacal light surface brightness [mag/arcsec**2]
- **SUN** (*bool, optional*) – Set to use solar spectrum (Not Implemented)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **czodi** – Zodiacal light photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**cezodi**(*q*, *X*, *T*, *lam*, *dlam*, *D*, *r*, *Fstar*, *Nez*, *Mezv*, *SUN=False*, *CIRC=False*)

Exozodiacal light count rate (Equation 18 from Robinson et al. 2016)

$$c_{\mathrm{ez}} = \pi q \mathcal{T} X^2 \frac{\lambda^4}{4hc\mathcal{R}} \left(\frac{1\mathrm{AU}}{r}\right)^2 \frac{F_{\mathrm{s},\lambda}(1\ \mathrm{AU})}{F_{\mathrm{s},V}(1\ \mathrm{AU})}$$
$$\times \frac{F_{\mathrm{s},V}(1\ \mathrm{AU})}{F_{\odot,V}(1\ \mathrm{AU})} N_{\mathrm{ez}} F_{0,V} 10^{-M_{\mathrm{ez},V}/2.5}$$

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **X** (*float*) – Size of photometric aperture (lambda/D)
- **T** (*float*) – System throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **D** (*float*) – Telescope diameter [m]
- **r** (*float*) – Planetary orbital semi-major axis [AU]
- **Fstar** (*array-like*) – Host star spectrum *at 1 au* (W/m**2/um)
- **Nez** (*float*) – Number of exozodis in exoplanetary disk
- **MezV** (*float*) – Exozodiacal light surface brightness [mag/arcsec**2]
- **SUN** (*bool, optional*) – Set to use solar spectrum (Not Implemented)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **cezodi** – Exozodiacal light photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**cspeck**(*q*, *T*, *C*, *lam*, *dlam*, *Fstar*, *D*)

Speckle count rate (Equation 19 from Robinson et al. 2016)

$$c_{\mathrm{sp}} = \pi q \mathcal{T} C \Delta\lambda F_{\mathrm{s},\lambda}(d) \frac{\lambda}{hc} \left(\frac{D}{2}\right)^2$$

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **T** (*float*) – System throughput
- **C** (*float, optional*) – Coronagraph design contrast
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **D** (*float*) – Telescope diameter [m]
- **Fstar** (*float or array-like*) – Host star spectrum at distance of planet (TOA) [W/m**2/um]

Returns **cspeck** – Speckle photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**cdark**(*De*, *X*, *lam*, *D*, *theta*, *DNhpix*, *IMAGE=False*, *CIRC=False*)

Dark current photon count rate

Parameters

- **De** (*float, optional*) – Dark current [counts/s]
- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **lam** (*float or array-like*) – Wavelength [um]
- **D** (*float*) – Telescope diameter [m]

- **theta** – Angular size of lenslet or pixel [arcsec**2]
- **DNHpix** (*float, optional*) – Number of horizontal/spatial pixels for dispersed spectrum
- **IMAGE** (*bool, optional*) – Set to indicate imaging mode (not IFS)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns Dark current photon count rate (s**-1)

Return type cdark

coronagraph.noise_routines.**cread**(*Re*, *X*, *lam*, *D*, *theta*, *DNhpix*, *Dtmax*, *IMAGE=False*, *CIRC=False*)

Read noise count rate (assuming detector has a maximum exposure time)

Parameters

- **Re** (*float or array-like*) – Read noise counts per pixel
- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **lam** (*float or array-like*) – Wavelength [um]
- **D** (*float*) – Telescope diameter [m]
- **theta** – Angular size of lenslet or pixel [arcsec**2]
- **Dtmax** (*float, optional*) – Detector maximum exposure time [hours]
- **IMAGE** (*bool, optional*) – Set to indicate imaging mode (not IFS)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **cread** – Read noise photon count rate (s**-1)

Return type float or array-like

coronagraph.noise_routines.**ccic**(*Rc*, *cscene*, *X*, *lam*, *D*, *theta*, *DNhpix*, *Dtmax*, *IMAGE=False*, *CIRC=False*)

Clock induced charge count rate

Parameters

- **Rc** (*float or array-like*) – Clock induced charge counts/pixel/photon
- **cscene** (*float or array-like*) – Photon count rate of brightest pixel in the scene [counts/s]
- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **lam** (*float or array-like*) – Wavelength [um]
- **D** (*float*) – Telescope diameter [m]
- **theta** – Angular size of lenslet or pixel [arcsec**2]
- **Dtmax** (*float, optional*) – Detector maximum exposure time [hours]
- **IMAGE** (*bool, optional*) – Set to indicate imaging mode (not IFS)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns Clock induced charge count rate [1/s]

Return type ccic

coronagraph.noise_routines.**f_airy**(*X*, *CIRC=False*)

Fraction of Airy power contained in square or circular aperture

Parameters

- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **f_airy** – Fraction of planetary light that falls within photometric aperture X*lambda/D

Return type float

coronagraph.noise_routines.**ctherm**(*q*, *X*, *T*, *lam*, *dlam*, *D*, *Tsys*, *emis*, *CIRC=False*)

Telescope thermal count rate

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **T** (*float*) – System throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **D** (*float*) – Telescope diameter [m]
- **Tsys** (*float*) – Telescope mirror temperature [K]
- **emis** (*float*) – Effective emissivity for the observing system (of order unity)
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **ctherm** – Telescope thermal photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**ctherm_earth**(*q*, *X*, *T*, *lam*, *dlam*, *D*, *Itherm*, *CIRC=False*)

Earth atmosphere thermal count rate

Parameters

- **q** (*float or array-like*) – Quantum efficiency
- **X** (*float, optional*) – Width of photometric aperture (* lambda / diam)
- **T** (*float*) – System throughput
- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]
- **D** (*float*) – Telescope diameter [m]
- **Itherm** (*float or array-like*) – Earth thermal intensity [W/m**2/um/sr]
- **CIRC** (*bool, optional*) – Set to use a circular aperture

Returns **cthe** – Earth atmosphere thermal photon count rate [1/s]

Return type float or array-like

coronagraph.noise_routines.**lambertPhaseFunction**(*alpha*)

Calculate the Lambertian Phase Function from the phase angle,

$$\Phi_{\mathrm{L}}(\alpha) = \frac{\sin\alpha + (\pi - \alpha)\cos\alpha}{\pi}$$

Parameters **alpha** (*float*) – Planet phase angle [deg]

Returns **Phi** – Lambertian phase function

Return type float

coronagraph.noise_routines.**construct_lam**(*lammin*, *lammax*, *Res=None*, *dlam=None*)
Construct a wavelength grid by specifying either a resolving power (*Res*) or a bandwidth (*dlam*)

Parameters

- **lammin** (*float*) – Minimum wavelength [microns]
- **lammax** (*float*) – Maximum wavelength [microns]
- **Res** (*float, optional*) – Resolving power (lambda / delta-lambda)
- **dlam** (*float, optional*) – Spectral element width for evenly spaced grid [microns]

Returns

- **lam** (*float or array-like*) – Wavelength [um]
- **dlam** (*float or array-like*) – Spectral element width [um]

coronagraph.noise_routines.**set_quantum_efficiency**(*lam*, *qe*, *NIR=False*, *qe_nir=0.9*, *vod=False*)
Set instrumental quantum efficiency

Parameters

- **lam** (*float or array-like*) – Wavelength [um]
- **qe** (*float*) – Detector quantum efficiency
- **NIR** (*bool, optional*) – Use near-IR detector proporties
- **q_nir** (*float, optional*) – NIR quantum efficiency
- **vod** (*bool, optional*) – "Valley of Death" red QE parameterization from Robinson et al. (2016)

Returns **q** – Wavelength-dependent instrumental quantum efficiency

Return type numpy.array

coronagraph.noise_routines.**set_dark_current**(*lam*, *De*, *lammax*, *Tdet*, *NIR=False*, *De_nir=0.001*)
Set dark current grid as a function of wavelength

Parameters

- **lam** (*array-like*) – Wavelength grid [microns]
- **De** (*float*) – Dark current count rate per pixel (s**-1)
- **lammax** (*float*) – Maximum wavelength
- **Tdet** (*float*) – Detector Temperature [K]
- **NIR** (*bool, optional*) – Use near-IR detector proporties
- **De_nir** (*float, optional*) – NIR minimum dark current count rate per pixel

Returns **De** – Dark current as a function of wavelength

Return type numpy.array

coronagraph.noise_routines.**set_read_noise**(*lam*, *Re*, *NIR=False*, *Re_nir=2.0*)
Set read noise grid as a function of wavelength

Parameters

- **lam** (*array-like*) – Wavelength grid [microns]

- **Re** (*float*) – Read noise counts per pixel (s**-1)
- **NIR** (*bool, optional*) – Use near-IR detector proporties
- **Re_nir** (*float, optional*) – NIR read noise counts per pixel

Returns **Re** – Read noise as a function of wavelength

Return type numpy.array

coronagraph.noise_routines.**set_lenslet**(*lam*, *lammin*, *diam*, *X*, *NIR=True*, *lammin_nir=1.0*)

Set the angular size of the lenslet

Parameters

- **lam** (*ndarray*) – Wavelength grid
- **lammin** (*float*) – Minimum wavelength
- **diam** (*float*) – Telescope Diameter [m]
- **X** (*float*) – Width of photometric aperture (*lambda/diam)
- **NIR** (*bool (optional)*) – Use near-IR detector proporties
- **lammin_nir** (*float (optional)*) – Wavelength min to use for NIR lenslet size

Returns **theta** – Angular size of lenslet

Return type numpy.array

coronagraph.noise_routines.**set_throughput**(*lam*, *Tput*, *diam*, *sep*, *IWA*, *OWA*, *lammin*, *FIX_OWA=False*, *SILENT=False*)

Set wavelength-dependent telescope throughput such that it is zero inside the IWA and outside the OWA.

Parameters

- **lam** (*ndarray*) – Wavelength grid
- **Tput** (*float*) – Throughput
- **diam** (*float*) – Telescope diameter [m]
- **sep** (*float*) – Planet-star separation in radians
- **IWA** (*float*) – Inner working angle
- **OWA** (*float*) – Outer working angle
- **lammin** (*float*) – Minimum wavelength
- **SILENT** (*bool, optional*) – Suppress printing

Returns **T** – Wavelength-dependent throughput

Return type numpy.array

coronagraph.noise_routines.**set_atmos_throughput**(*lam*, *dlam*, *convolve*, *plot=False*)

Use pre-computed Earth atmospheric transmission to set throughput term for radiation through the atmosphere

Parameters

- **lam** (*ndarray*) – Wavelength grid
- **dlam** (*ndarray*) – Wavelength bin width grid
- **convolve** (*func*) – Function used to degrade/downbin spectrum

Returns **Tatmos** – Atmospheric throughput as a function of wavelength

Return type numpy.array

coronagraph.noise_routines.**get_sky_flux**()
Get the spectral flux density from the sky viewed at a ground-based telescope an an average night. This calculation comes from ESO SKYCALC and includes contributions from molecular emission of lower atmosphere, emission lines of upper atmosphere, and airglow/residual continuum, but neglects scattered moonlight, starlight, and zodi.

Returns

- **lam_sky** (*numpy.array*) – Wavelength grid [microns]
- **flux_sky** (*numpy.array*) – Flux from the sky [W/m^2/um]

coronagraph.noise_routines.**exptime_element**(*lam*, *cp*, *cn*, *wantsnr*)
Calculate the exposure time (in hours) to get a specified signal-to-noise

Parameters

- **lam** (*ndarray*) – Wavelength grid
- **cp** (*ndarray*) – Planetary photon count rate [s**-1]
- **cn** (*ndarray*) – Noise photon count rate [s**-1]
- **wantsnr** (*float*) – Signal-to-noise required in each spectral element

Returns **DtSNR** – Exposure time necessary to get specified SNR [hours]

Return type ndarray

coronagraph.noise_routines.**planck**(*temp*, *wav*)
Planck blackbody function

Parameters

- **temp** (*float or array-like*) – Temperature [K]
- **wav** (*float or array-like*) – Wavelength [microns]

Returns

Return type B_lambda [W/m^2/um/sr]

1.5.5 Degrade Spectra

Methods for degrading high-resolution spectra to lower resolution.

coronagraph.degrade_spec.**downbin_spec**(*specHR*, *lamHR*, *lamLR*, *dlam=None*)
Re-bin spectum to lower resolution using `scipy.binned_statistic` with `statistic = 'mean'`. This is a "top-hat" convolution.

Parameters

- **specHR** (*array-like*) – Spectrum to be degraded
- **lamHR** (*array-like*) – High-res wavelength grid
- **lamLR** (*array-like*) – Low-res wavelength grid
- **dlam** (*array-like, optional*) – Low-res wavelength width grid

Returns **specLR** – Low-res spectrum

Return type `numpy.ndarray`

coronagraph.degrade_spec.**downbin_spec_err**(*specHR*, *errHR*, *lamHR*, *lamLR*, *dlam=None*)
Re-bin spectum and errors to lower resolution using `scipy.binned_statistic`. This function calculates the noise weighted mean of the points within a bin such that $\sqrt{\sum_i \mathrm{SNR}_i}$ within each i bin is preserved.

Parameters

- **specHR** (*array-like*) – Spectrum to be degraded
- **errHR** (*array-like*) – One sigma errors of high-res spectrum
- **lamHR** (*array-like*) – High-res wavelength grid
- **lamLR** (*array-like*) – Low-res wavelength grid
- **dlam** (*array-like, optional*) – Low-res wavelength width grid

Returns

- **specLR** (`numpy.ndarray`) – Low-res spectrum
- **errLR** (`numpy.ndarray`) – Low-res spectrum one sigma errors

coronagraph.degrade_spec.**degrade_spec**(*specHR*, *lamHR*, *lamLR*, *dlam=None*)
Degrade high-resolution spectrum to lower resolution (DEPRECIATED)

> **Warning:** This method is known to return incorrect results at relatively high spectral resolution and has been depreciated within the `coronagraph` model. Please use *downbin_spec()* instead.

Parameters

- **specHR** (*array-like*) – Spectrum to be degraded
- **lamHR** (*array-like*) – High-res wavelength grid
- **lamLR** (*array-like*) – Low-res wavelength grid
- **dlam** (*array-like, optional*) – Low-res wavelength width grid

Returns **specLO** – Low-res spectrum

Return type `numpy.ndarray`

1.5.6 Filter Photometry

The following *Filter* and *Wheel* classes can be used to simulate coronagraph observations in imaging mode.

class coronagraph.imager.**Filter**(*name=None*, *bandcenter=None*, *FWHM=None*, *wl=None*, *response=None*, *notes=''*)
Filter for telescope imaging mode.

Parameters

- **name** (*string*) – Name of filter
- **bandcenter** (*float*) – Wavelength at bandcenter (um)
- **FWHM** (*float*) – Full-width as half-maximum (um)
- **wl** (*array*) – Wavelength grid for response function (um)
- **response** (*array*) – Filter response function
- **notes** (*string*) – Notes of filter, e.g. 'Johnson-Cousins'

class coronagraph.imager.**Wheel**
Filter Wheel. Contains different filters as attributes.

add_new_filter(*filt*, *name='new_filter'*)
Adds new filter to wheel

Parameters

- **filt** (Filter) – New Filter object to be added to wheel
- **name** (*string (optional)*) – Name to give new filter attribute

plot(*ax=None*, *ylim=None*)
Plot the filter response functions

Parameters **ax** (*matplotlib.axis (optional)*) – Axis instance to plot on

Returns **ax** – Axis with plot

Return type matplotlib.axis

Note: Only returns an axis if an axis was not provided.

coronagraph.imager.**read_jc**()
Read and parse the Johnson-Cousins filter files.

Returns

- **filters** (*numpy.ndarray*) – Array of filter response functions
- **filter_names** (*list*) – List of string names for the filters
- **bandcenters** (*numpy.array*) – Wavelength bandcenters for the filters [microns]
- **FWHM** (*numpy.array*) – Full-width at half max for the filters

class coronagraph.imager.**johnson_cousins**
Instantiate a filter *Wheel* with the Johnson-Cousins filters (U, B, V, R, I).

Example

```
>>> jc = cg.imager.johnson_cousins()
>>> jc.plot(ylim = (0.0, 1.2))
```

coronagraph.imager.**read_landsat**()
Read and parse the LANDSAT filter files.

Returns

- **wl** (*list*) – List of wavelength grids for each filter [microns]
- **response** (*list*) – Filter responses for each filter
- **LANDSAT_names** (*list*) – Names of each LANDSAT filter
- **FWHM** (*numpy.array*) – Full-width at half max for the filters
- **bandcenters** (*numpy.array*) – Wavelength bandcenters for the filters [microns]

class coronagraph.imager.**landsat**
Instantiate a filter *Wheel* with the LANDSAT filters.

coronagraph.imager.**read_jc2**()
Read and parse the Johnson-Cousins Bessel filter files.

Returns

- **filters** (*numpy.ndarray*) – Array of filter response functions
- **filter_names** (*list*) – List of string names for the filters
- **bandcenters** (*numpy.array*) – Wavelength bandcenters for the filters [microns]
- **FWHM** (*numpy.array*) – Full-width at half max for the filters

class coronagraph.imager.**johnson_cousins2**
Instantiate a filter *Wheel* with the Johnson-Cousins Bessel filters (U, B, V, R, I).

1.5.7 Simulating Transmission and Emission Spectroscopy

Simulate exoplanet transmission and/or emission spectroscopy without using the coronagraph routines. This uses the same telesope and detector parameters as the coronagraph model, but does not suppress the star's light. As a result, stellar photons dominate the noise budget.

For transmission spectroscopy calculations use *TransitNoise*, and for emission spectroscopy use *EclipseNoise*. You may also get an example transmission and emission spectrum of the Earth by calling *get_earth_trans_spectrum()*.

class coronagraph.transits.**EclipseNoise**(*tdur=3432.0*, *telescope=<coronagraph.teleplanstar.Telescope object>*, *planet=<coronagraph.teleplanstar.Planet object>*, *star=<coronagraph.teleplanstar.Star object>*, *ntran=1*, *nout=1*, *wantsnr=1000.0*, *NIR=True*, *THERMAL=True*, *GROUND=False*, *vod=False*, *IMAGE=False*)
Simulate exoplanet secondary eclipse emission spectroscopy with a next-generation telescope.

Parameters

- **telescope** (Telescope) – Initialized object containing Telescope parameters
- **planet** (Planet) – Initialized object containing Planet parameters
- **star** (Star) – Initialized object containing Star parameters
- **tdur** (*float*) – Transit duration [s]
- **ntran** (*float*) – Number of transits/eclipses
- **nout** (*float*) – Number of out-of-eclipse transit durations to observe
- **wantsnr** (*float, optional*) – Desired signal-to-noise ratio in each pixel
- **FIX_OWA** (*bool, optional*) – Set to fix OWA at OWA*lammin/D, as would occur if lenslet array is limiting the OWA
- **COMPUTE_LAM** (*bool, optional*) – Set to compute lo-res wavelength grid, otherwise the grid input as variable lam is used
- **SILENT** (*bool, optional*) – Set to suppress print statements
- **NIR** (*bool, optional*) – Re-adjusts pixel size in NIR, as would occur if a second instrument was designed to handle the NIR
- **THERMAL** (*bool, optional*) – Set to compute thermal photon counts due to telescope temperature

- **GROUND** (*bool, optional*) – Set to simulate ground-based observations through atmosphere
- **vod** (*bool, optional*) – "Valley of Death" red QE parameterization from Robinson et al. (2016)

run_count_rates(*lamhr=None*, *Fphr=None*, *Fshr=None*)

Calculate the photon count rates and signal to noise on a secondary eclipse spectrum observation

Parameters

- **lamhr** (*numpy.ndarray*) – Wavelength [mum]
- **Fphr** (*numpy.ndarray*) – Dayside exoplanet TOA flux spectrum [W/m^2/mu]
- **Fshr** (*numpy.ndarray*) – Stellar flux incident at the planet's TOA [W/m^2/mu]
- **run_count_rates() creates the following attributes for** (*Calling*) –
- **EclipseNoise instance** (*the*) –

Variables

- **lamhr** (*array*) – Wavelength [mum]
- **Fphr** (*array*) – Dayside exoplanet TOA flux spectrum [W/m^2/mu]
- **Fshr** (*array*) – Stellar flux incident at the planet's TOA [W/m^2/mu]
- **cs** (*array*) – Stellar photon count rate [photons/s]
- **cback** (*array*) – Background photon count rate [photons/s]
- **cz** (*array*) – Zodi photon count rate [photons/s]
- **cez** (*array*) – Exo-zodi photon count rate [photons/s]
- **cth** (*array*) – Thermal photon count rate [photons/s]
- **cD** (*array*) – Dark current photon count rate [photons/s]
- **cR** (*array*) – Read noise photon count rate [photons/s]
- **cmiss** (*array*) – Occulted stellar photon count rate [photons/s]
- **SNR1** (*array*) – S/N for one eclipse
- **SNRn** (*array*) – S/N for ntran eclipses
- **tSNR** (*array*) – Exposure time to wantsnr [s]
- **nSNR** (*array*) – Number of eclipses to wantsnr
- **lam** (*array*) – Observed wavelength grid [mum]
- **dlam** (*array*) – Observed wavelength grid widths [mum]
- **FpFslr** (*array*) – Low-res planet/star flux ratio
- **FpFshr** (*array*) – High-res planetr/star flux ratio

make_fake_data()

Make a fake dataset by sampling from a Gaussian.

Variables

- **SNRn** (*array*) – S/N in ntran eclipses
- **obs** (*array*) – Observed emission specrum with noise

- **sig** (*array*) – Observed uncertainties on emission spectrum

recalc_wantsnr(*wantsnr=None*)
Recalculate the time and number of eclipses required to achieve a user specified SNR via *wantsnr*.

Variables

- **tSNR** (*array*) – Exposure time to `wantsnr` [s]
- **nSNR** (*array*) – Number of eclipses to `wantsnr`

plot_spectrum(*SNR_threshold=0.0*, *Nsig=None*, *ax0=None*, *err_kws={'alpha': 1, 'c': 'k', 'fmt': '.'}*, *plot_kws={'alpha': 0.5, 'c': 'C4', 'lw': 1.0}*, *draw_box=True*)
Plot noised emission spectrum.

Parameters

- **SNR_threshold** (*float*) – Threshold SNR below which do not plot
- **Nsig** (*float*) – Number of standard deviations about median observed points to set yaxis limits
- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **err_kws** (*dic*) – Keyword arguments for *errorbar*
- **plot_kws** (*dic*) – Keyword arguments for *plot*
- **draw_box** (*bool*) – Draw important quantities in a box?

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

---

plot_SNRn(*ax0=None*, *plot_kws={'ls': 'steps-mid'}*)
Plot the S/N on the Eclipse Depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

---

plot_ntran_to_wantsnr(*ax0=None*, *plot_kws={'alpha': 1.0, 'ls': 'steps-mid'}*)
Plot the number of eclipses to get a SNR on the eclipse depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

plot_time_to_wantsnr(*ax0=None*, *plot_kws={'alpha': 1.0, 'ls': 'steps-mid'}*)
Plot the time to get a SNR on the eclipse depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

plot_count_rates(*ax0=None*)
Plot the photon count rate for all sources.

Parameters **ax0** (*matplotlib.axes*) – Optional axis to provide

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

class coronagraph.transits.**TransitNoise**(*tdur=3432.0*, *telescope=<coronagraph.teleplanstar.Telescope object>*, *planet=<coronagraph.teleplanstar.Planet object>*, *star=<coronagraph.teleplanstar.Star object>*, *ntran=1*, *nout=1*, *wantsnr=1000.0*, *NIR=True*, *THERMAL=True*, *GROUND=False*, *vod=False*, *IMAGE=False*)
Simulate exoplanet transit transmission spectroscopy with a next-generation telescope.

Parameters

- **telescope** (`Telescope`) – Initialized object containing `Telescope` parameters
- **planet** (`Planet`) – Initialized object containing `Planet` parameters
- **star** (`Star`) – Initialized object containing `Star` parameters
- **tdur** (*float*) – Transit duration [s]
- **ntran** (*float*) – Number of transits
- **nout** (*float*) – Number of out-of-transit transit durations to observe
- **wantsnr** (*float, optional*) – Desired signal-to-noise ratio in each pixel

- **FIX_OWA** (*bool, optional*) – Set to fix OWA at `OWA*lammin/D`, as would occur if lenslet array is limiting the OWA
- **COMPUTE_LAM** (*bool, optional*) – Set to compute lo-res wavelength grid, otherwise the grid input as variable `lam` is used
- **SILENT** (*bool, optional*) – Set to suppress print statements
- **NIR** (*bool, optional*) – Re-adjusts pixel size in NIR, as would occur if a second instrument was designed to handle the NIR
- **THERMAL** (*bool, optional*) – Set to compute thermal photon counts due to telescope temperature
- **GROUND** (*bool, optional*) – Set to simulate ground-based observations through atmosphere
- **vod** (*bool, optional*) – "Valley of Death" red QE parameterization from Robinson et al. (2016)

run_count_rates(*lamhr=None*, *tdhr=None*, *Fshr=None*)

Calculate the photon count rates and signal to noise on a transmission spectrum observation

Parameters

- **lamhr** (*numpy.ndarray*) – Wavelength [mum]
- **tdhr** (*numpy.ndarray*) – Transit Depth $(Rp/Rs)^2$
- **Fshr** (*numpy.ndarray*) – Flux density incident at the planet's TOA [W/m^2/mu]
- **run_count_rates() creates the following attributes for** (*Calling*) –
- **TransitNoise instance** (*the*) –

Variables

- **lamhr** (*array*) – Wavelength [mum]
- **tdhr** (*array*) – Transit Depth $(Rp/Rs)^2$
- **Fshr** (*array*) – Flux density incident at the planet's TOA [W/m^2/mu]
- **cs** (*array*) – Stellar photon count rate [photons/s]
- **cback** (*array*) – Background photon count rate [photons/s]
- **cz** (*array*) – Zodi photon count rate [photons/s]
- **cez** (*array*) – Exo-zodi photon count rate [photons/s]
- **cth** (*array*) – Thermal photon count rate [photons/s]
- **cD** (*array*) – Dark current photon count rate [photons/s]
- **cR** (*array*) – Read noise photon count rate [photons/s]
- **cmiss** (*array*) – Occulted stellar photon count rate [photons/s]
- **SNR1** (*array*) – S/N for one transit
- **SNRn** (*array*) – S/N for `ntran` transits
- **tSNR** (*array*) – Exposure time to `wantsnr` [s]
- **nSNR** (*array*) – Number of transits to `wantsnr`
- **lam** (*array*) – Observed wavelength grid [mum]

- **dlam** (*array*) – Observed wavelength grid widths [mum]
- **RpRs2** (*array*) – Low-res transit depth

make_fake_data()
Make a fake dataset by sampling from a Gaussian.

Variables

- **SNRn** (*array*) – S/N in ntran transits
- **obs** (*array*) – Observed transit depth with noise
- **sig** (*array*) – Observed uncertainties on transit depth

recalc_wantsnr(*wantsnr=None*)
Recalculate the time and number of transits required to achieve a user specified SNR via *wantsnr*.

Variables

- **tSNR** (*array*) – Exposure time to wantsnr [s]
- **nSNR** (*array*) – Number of transits to wantsnr

plot_spectrum(*SNR_threshold=1.0*, *Nsig=6.0*, *ax0=None*, *err_kws={'alpha': 1*, *'c': 'k'*, *'fmt': '.'}*, *plot_kws={'alpha': 0.5*, *'c': 'C4'*, *'lw': 1.0}*, *draw_box=True*)
Plot noised transmission spectrum.

Parameters

- **SNR_threshold** (*float*) – Threshold SNR below which do not plot
- **Nsig** (*float*) – Number of standard deviations about median observed points to set yaxis limits
- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **err_kws** (*dic*) – Keyword arguments for *errorbar*
- **plot_kws** (*dic*) – Keyword arguments for *plot*
- **draw_box** (*bool*) – Draw important quantities in a box?

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is ax0 is None

---

plot_SNRn(*ax0=None*, *plot_kws={'ls': 'steps-mid'}*)
Plot the S/N on the Transit Depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

plot_ntran_to_wantsnr(*ax0=None*, *plot_kws={'alpha': 1.0, 'ls': 'steps-mid'}*)
Plot the number of transits to get a SNR on the transit depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

plot_time_to_wantsnr(*ax0=None*, *plot_kws={'alpha': 1.0, 'ls': 'steps-mid'}*)
Plot the time to get a SNR on the transit depth as a function of wavelength.

Parameters

- **ax0** (*matplotlib.axes*) – Optional axis to provide
- **plot_kws** (*dic*) – Keyword arguments for *plot*

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

plot_count_rates(*ax0=None*)
Plot the photon count rate for all sources.

Parameters **ax0** (*matplotlib.axes*) – Optional axis to provide

Returns

- **fig** (*matplotlib.figure.Figure*) – Returns a figure if *ax0* is *None*
- **ax** (*matplotlib.axes*) – Returns an axis if *ax0* is *None*

Note: Only returns *fig* and *ax* is `ax0 is None`

coronagraph.transits.**get_earth_trans_spectrum**()
Get the transmission spectrum of the Earth around the Sun.

Returns

- **lam** (*numpy.ndarray*) – Wavelength grid [um]
- **tdepth** (*numpy.ndarray*) – Transit depth (Rp/Rs)^2
- **fplan** (*numpy.ndarray*) – TOA planet flux [W/m^2/um]

- **fstar** (*numpy.ndarray*) – Stellar flux at planet [W/m^2/um]

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

C

S

INDEX

T

W