trappist_project

December 4, 2017

1 Simulating Observing Within the TRAPPIST-1 System

The TRAPPIST-1 exoplanet system was originally found to have 3 short-period transiting planets orbiting a cool M-dwarf star in May 2015. Later, in 2017, the system was shown to have at least 7 planets of which at least 3 (TRAPPIST-1 e, f, g) are in the Goldilocks zone for liquid surface water. Morley et. al 2017 generated a grid of models for these planets among many for different assumptions of surface pressures, bond albedos, and atmosphere compositions.

Here I assume the observer planet is TRAPPIST-1 f, with an Earth-like atmosphere, $A_B = 0.3$, and $P_{\text{surf}} = 1$ bar. We are observing our immediate neighbor: TRAPPIST-1 g with a Titan-like atmosphere, $A_B = 0$, and $P_{\text{surf}} = 0.001$ bar. The geometry is such that we are observing this planet at night through our own atmosphere. Since the target has no albedo and will not reflect the light of our host star, we do not include it in our analysis.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from spectroscopy import planck_flux
        from astropy import units as u
        from scipy.optimize import fmin as minimize
        from astropy import constants as const
        from ipywidgets import *
        from astropy.convolution import convolve, Gaussian1DKernel
        from IPython.display import set_matplotlib_formats
        set_matplotlib_formats('pdf', 'png')
        plt.rcParams['savefig.dpi'] = 150
```

```
In [2]: inner_planet = np.loadtxt('trappist1f_psurf1.0_alb0.3_chem_earth.spec',
                                  skiprows=4)
        outer_planet = np.loadtxt('trappist1g_psurf0.001_alb0.0_chem_titan.spec',
                                  skiprows=4)
        inner_mask = np.where((inner_planet[:,0]> 0.8)
                              & (inner_planet[:,0] < 30.0))
        outer_mask = np.where((outer_planet[:,0]> 0.8)
                              & (outer_planet[:,0] < 30.0))

        wien_constant = 2897.77729 #micron K
        inner_lambda_max = inner_planet[:,0][np.argmax(inner_planet[:,1])]
        inner_temp_surf = wien_constant/inner_lambda_max
        outer_lambda_max = outer_planet[:,0][np.argmax(outer_planet[:,1])]
        outer_temp_surf = wien_constant/outer_lambda_max

        inner_planck = planck_flux(inner_planet[:,0]*u.m/(10.0**6),
                                   inner_temp_surf*u.K)
        inner_factor = (np.max(inner_planet[:, 1])/np.max(inner_planck))
        inner_planck = inner_planck*inner_factor
        inner_res = minimize(lambda x: np.sum(np.abs((x[1]*planck_flux(inner_planet[:,0]*u.mic
                                                                       x[0]*u.K)).value
                                                     - inner_planet[:,1])),
                             [inner_temp_surf, inner_factor.value])
        inner_planck = inner_res[1]*planck_flux(inner_planet[:,0]*u.micron,
                                                inner_res[0]*u.K)

        outer_planck = planck_flux(outer_planet[:,0]*u.m/(10.0**6),
                                   outer_temp_surf*u.K)
        outer_factor = (np.max(outer_planet[:, 1])/np.max(outer_planck)).value
        outer_planck = outer_planck*outer_factor
        outer_res = minimize(lambda x: np.sum(np.abs((x[1]*planck_flux(outer_planet[:,0]*u.mic
                                                                       x[0]*u.K)).value
                                                     - outer_planet[:,1])),
                             [outer_temp_surf, outer_factor])

        outer_planck = planck_flux(outer_planet[:,0]*u.micron, outer_res[0]*u.K)*outer_res[1]
```

```
Optimization terminated successfully.
         Current function value: 578147950.439309
         Iterations: 106
         Function evaluations: 202
Optimization terminated successfully.
         Current function value: 25955690.604147
         Iterations: 78
         Function evaluations: 148
```

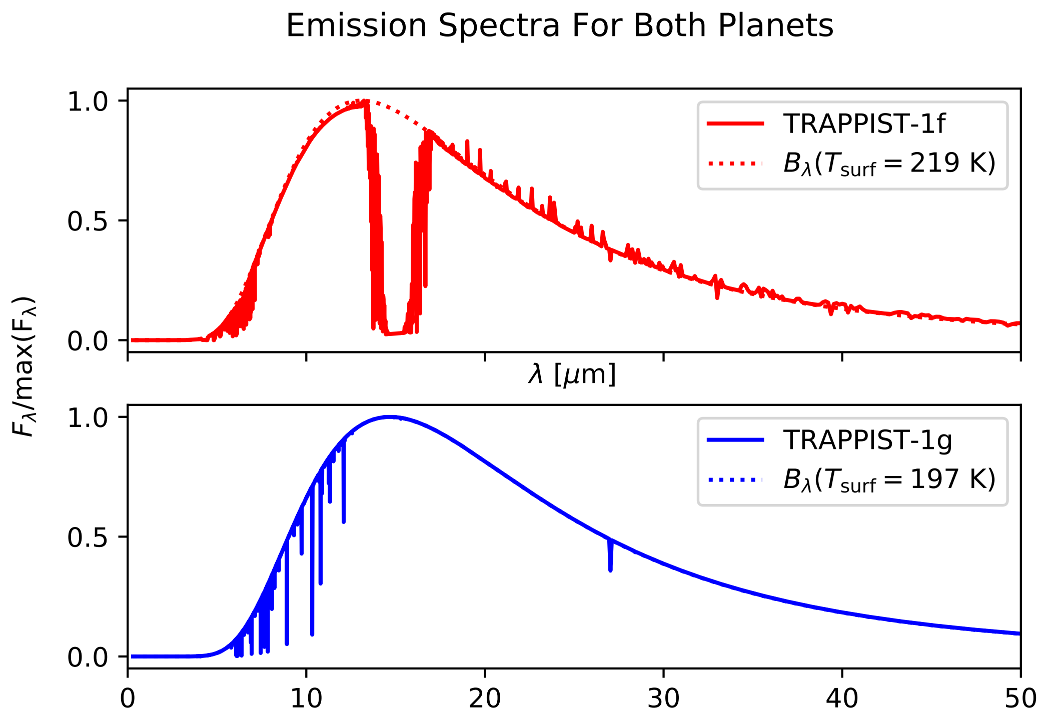
```
In [3]: outer_ang_size = ((1.127*const.R_earth)/(0.008*const.au)).decompose()
        outer_angular_area = 2.0*np.pi*(1.0 - np.cos(outer_ang_size.value))

        inner_resids = np.abs(inner_planet[:,1] - inner_planck.value)
        inner_max_absorption_loc = np.argmax(inner_resids)
        inner_tau_max = -np.log(inner_planet[:,1][inner_max_absorption_loc]
                                /inner_planck.value[inner_max_absorption_loc])

        def get_eq_skin(star_temp, albedo, star_rad, planet_orb):
            eq_temp = star_temp*((1.0 - albedo)**0.25)*np.sqrt((star_rad/(2.0*planet_orb)).dec
            return eq_temp, eq_temp/(2.0**0.25)

        inner_eq, inner_skin = get_eq_skin(2550.0, 0.3, 0.114*const.R_sun, 0.0451*const.au)
```

```
In [4]: fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
        ax1.plot(inner_planet[:, 0], inner_planet[:, 1]/np.max(inner_planet[:, 1]),
                 '-r',label="TRAPPIST-1f")
        ax2.plot(outer_planet[:, 0], outer_planet[:, 1]/np.max(outer_planet[:, 1]),
                 '-b',label="TRAPPIST-1g")
        ax1.plot(inner_planet[:,0], inner_planck.value/np.max(inner_planck.value),
                 ':r',
                 label=r'$B_{\lambda}(T_{\mathrm{surf}} =$'
                 + str(int(inner_temp_surf))
                 + ' K)')
        ax2.plot(outer_planet[:,0], outer_planck.value/np.max(outer_planck.value),
                 ":b",
                 label=r'$B_{\lambda}(T_{\mathrm{surf}} = $'
                 + str(int(outer_temp_surf))
                 + ' K)')
        ax1.set_xlabel(r'$\lambda$ [$\mu$m]')
        ax2.set_ylabel(r'$F_{\lambda}/\mathrm{max(F_\lambda)}$')
        ax2.yaxis.set_label_coords(-0.1, 1.15)
        ax1.legend()
        ax1.set_xlim(0.0, 50.0)
        ax2.legend()
        plt.suptitle('Emission Spectra For Both Planets')
        plt.show()
```



Emission Spectrum:

$$F_\lambda = \left(B_\lambda(T_{\text{surf}})e^{-\tau_\lambda} + \int_0^{\tau_\lambda} B_\lambda(T_{\tau'_\lambda})e^{-\tau'_\lambda}d\tau'_\lambda \right) \times 2\pi\,(1 - \cos\theta)$$

where θ is the angular radius of the object. Plotted above are the emission spectra for both the observer and target planet along with the spectra of blackbodies at the respective surface temperatures of both planets.

Conceptually, looking from outside the atmosphere, where τ_λ is the optical depth from the surface to the top of the atmosphere, the flux we see is a combination of the surface radiating upwards (decayed by $e^{-\tau_\lambda}$) and individual slabs of the atmosphere radiating at their own temperatures (with their own corresponding decay).

The details of these components do not matter at all for our target planet since the model emission spectrum provided by Morley et al. 2017 is all we care about. However, to incorporate the atmosphere of our own observer planet, we must know what our atmosphere radiates towards our telescope, picking out the integral term from the model emission spectrum provided.

Fortunately, our atmosphere is transparent at many wavelengths, allowing us to see the overall blackbody shape corresponding to the surface temperature which we can solve for. We also have a very deep absorption feature around 15 μm which will help us pick out the integral component we care for.

Finding $F_{\lambda_{min}}$ and comparing it to what $B_{\lambda_{min}}(T_{\text{surf}})$ should be at that wavelength, we can solve for the maximum τ_λ. Then, by assuming T_{τ_λ} varies linearly with τ_λ from T_{surf} to T_{skin}, and using Taylor expansions to approximate the integral $\int_0^{\tau_\lambda} B_\lambda(T_{\tau'_\lambda})e^{-\tau'_\lambda}d\tau'_\lambda$, we can solve for $\tau_\lambda(\lambda)$, then use that to find only the integral component. Assuming the atmosphere radiates isotropically, this is the specific intensity that radiates down towards our telescope.

```
In [5]: def find_optical_depth(tau_max, flux, wavelength,
                               surf_temp, eq_temp, skin_temp):
            temp_slope = u.K*(surf_temp - skin_temp)/(tau_max)
            m = temp_slope
            temp_intercept = skin_temp*u.K
            b = temp_intercept
            exp_factor = const.h*const.c/const.k_B
            test_taus = np.linspace(0.0, tau_max, 1000)
            final_taus = np.empty_like(wavelength)
            for i in range(0, len(wavelength)):
                wave = wavelength[i]*u.micron
                surf_term = planck_flux(wave, surf_temp*u.K)*np.exp(-test_taus)
                common_factor = 2.0*const.h*(const.c**2.0)/(wave**5.0)
                other_exp = (exp_factor/(wave)).decompose().value
                a = (const.h*const.c/(const.k_B*wave)).decompose()
                term_one = (test_taus**2.0)/(2.0*(np.exp(a/b) - 1.0))
                term_two = ((a*m*np.exp(a/b))*
                            (test_taus**3.0)/(6.0*(b**2.0)*
                                              (np.exp(a/b)
                                               - 1.0)**2.0))
                term_three = ((a*(m**2.0)*np.exp(a/b))*
                            (test_taus**4.0)/(24.0*(b**4.0)*
                                              (np.exp(a/b)
                                               - 1.0)**3.0))
                term_three = term_three*(a*np.exp(a/b)
                                         - 2.0*b*np.exp(a/b)
                                         + a + 2.0*b)
                test_integral = (surf_term
                                 + common_factor*(term_one
                                                  + term_two + term_three))
                diff = np.abs(flux[i]*(u.W/(u.m**3.0))
                              - 0.05738*test_integral.to(u.W/(u.m**3.0)))
                final_taus[i] = test_taus[np.argmin(diff)]
            return final_taus

        low = 0
        high = len(inner_planet[:,0])
        inner_tau_lambda = find_optical_depth(inner_tau_max,
                                              inner_planet[low:high,1],
                                              inner_planet[low:high,0],
                                              inner_temp_surf,
                                              inner_eq,
                                              inner_skin)
        inner_tau_lambda[np.where(inner_planet[:,0] > 20.0)] = 0.0

/home/gmduvvuri/miniconda3/envs/astroconda/lib/python3.5/site-packages/astropy/units/quantity.
  *arrays, **kwargs)
/home/gmduvvuri/miniconda3/envs/astroconda/lib/python3.5/site-packages/astropy/units/quantity.
  *arrays, **kwargs)
```

```
In [6]: def get_background_integral(wavelength, tau_lambdas, surf_temp, skin_temp):
            tau_max = np.max(tau_lambdas)
            temp_slope = u.K*(surf_temp - skin_temp)/(tau_max)
            m = temp_slope
            temp_intercept = skin_temp*u.K
            b = temp_intercept
            exp_factor = const.h*const.c/const.k_B
            integral_numbers = np.empty_like(wavelength)
            for i in range(0, len(wavelength)):
                wave = wavelength[i]*u.micron
                a = (const.h*const.c/(const.k_B*wave)).decompose()
                if tau_lambdas[i] > 0.00001:
                    tau_arr = np.linspace(0.0, tau_lambdas[i], 1000)
                    temp_arr = m*tau_arr + b
                    integrand = ((planck_flux(wave, temp_arr).to(u.W/((u.m**2.0)*u.micron)))/n
                    integral_numbers[i] = np.trapz(integrand.value, tau_arr)
                else:
                    integral_numbers[i] = 0.0
            return integral_numbers*((u.W/((u.m**2.0)*u.micron))
        inner_background_integral = get_background_integral(inner_planet[:,0], inner_tau_lambda
```

The model provided must be consistent with the spectrograph we assume, so we convolve it with a Gaussian kernel to match the resolution we assume, and then multiplied by the throughput of the telescope (f, letting through anywhere between 0% and 100% of the light). The flux we observe from our target is this convolved emission spectrum of TRAPPIST-1 g modified by the angular size of the target, the collecting area of our telescope (A), and the exposure time (t):

$$\text{Signal} = (F_\lambda * G(R \times \lambda_{\text{mean}})) \times \frac{\pi R_p^2}{4\pi d^2} \times A \times t \times f$$

where R_p is the radius of TRAPPIST-1 g and d is the distance between TRAPPIST-1 f and g, which we assume is simply the difference between their orbital distances (TRAPPIST-1 g is in perfect opposition and both planets are on perfectly circular orbits).

The noise is a combination of the Poisson noise of our signal, the Poisson noise of the flux from our own atmosphere which we circuitously (and only loosely approximately) solved for above, and white noise due to instrumental limitations (σ^2):

$$\text{Noise} = \sqrt{\text{Signal} + \left(\int_0^{\tau_\lambda} B_\lambda(T_{\tau'_\lambda}) e^{-\tau'_\lambda} d\tau'_\lambda \times f \times A \times t \times 2\pi\,(1 - \cos\theta)\right) + \sigma^2}$$

Then the SNR is simply:

$$\text{SNR} = \frac{\text{Signal}}{\text{Noise}}$$

For the purposes of our simulation, the parameters we can vary are:

$$R, f, t, A, \sigma^2$$

Of these, A we vary by changing the diameter of the telescope, and use $\log\sigma^2$ and $\log R$ instead of the values directly. For a wavelength range between λ_{min} and λ_{max} we can show the SNR as a function of wavelength and the true SNR is a mean over this function. Below, we allow these parameters to be varied and show the mean SNR integrated over the range specified using an interactive widget.

```
In [7]: def get_outer_flux(throughput, wave_min, wave_max, resolution):
            outer_planet_flux_received = throughput*outer_planet[:,1]*(outer_ang_size**2.0)/4.
            width = (np.mean([wave_min, wave_max])/resolution)
            d_lambda = np.mean(np.diff(outer_planet[:,0]))
            kernel = Gaussian1DKernel(-d_lambda/width)
            outer_planet_flux_observed = convolve(outer_planet_flux_received, kernel)
            outer_flux = outer_planet_flux_observed*(u.W/(u.m**3.0))
            outer_flux = outer_flux.to(u.W/(u.micron*(u.m**2.0)))
            return outer_flux

        def get_signal(flux, wavelength, area, time):
            return (flux*area*time).decompose()
```

```
def get_noise(flux, wavelength, background_integral, area,
             time, white_noise_variance, signal, throughput):
    return np.sqrt(signal.value*wavelength.value
                   + (background_integral*outer_angular_area
                      *throughput*area*time*wavelength.value).decompose().value
                   + white_noise_variance)

def get_snr(signal, noise):
    return signal/noise

def get_snr_integrated(snr, wavelength, wave_min, wave_max):
    mask = np.where((wavelength.value < wave_max)
                    & (wavelength.value >= wave_min))
    return np.mean((wavelength[mask].value*snr[mask].value)
                   /np.mean(wavelength[mask].value))

def calc_snr(area, time, white_noise_variance,
             throughput, resolution, wave_low, wave_high):
    outer_flux = get_outer_flux(throughput, wave_low, wave_high, resolution)
    signal = get_signal(outer_flux, outer_planet[:,0]*u.micron, area, time)
    noise = get_noise(outer_flux, outer_planet[:,0]*u.micron,
                      inner_background_integral, area, time,
                      white_noise_variance, signal, throughput)
    snr = get_snr(signal, noise)
    return snr

def plot_snr(telescopeD, time, log_variance,
             throughput, log_resolution, wave_low, wave_high):
    radius = telescopeD/2.0
    area = np.pi*((radius*u.m)**2.0)
    time = time*u.s
    white_noise_variance = 10.0**log_variance
    resolution = 10.0**log_resolution
    snr = calc_snr(area, time, white_noise_variance,
                   throughput, resolution, wave_low, wave_high)
    integrated_snr = get_snr_integrated(snr,
                                        outer_planet[:,0]*u.micron,
                                        wave_low, wave_high)
    plt.plot(outer_planet[:,0], snr)
    plt.title(r'$\langle \mathrm{SNR}\rangle =$ ' + str(integrated_snr))
    plt.xlabel(r'$\lambda$ [$\mu$m]')
    plt.ylabel(r'$\mathrm{SNR}_\lambda$')
    plt.xlim(wave_low, wave_high)
    plt.show()

interact(plot_snr, telescopeD=(0.1, 20.0, 0.1),
         time=(0.0, 600.0, 20.0), log_variance=(0.0, 10.0, 1.0),
         throughput=(0.0, 1.0, 0.1), log_resolution=(0.0, 5.0, 0.5),
                  wave_low = (0.0, 10.0, 1.0), wave_high=(20.0, 50.0, 1.0))
```

```
interactive(children=(FloatSlider(value=10.0, description='telescopeD', max=20.0, min=0.1), Fl
```

```
Out[7]: <function __main__.plot_snr>
```

"telescopeD": Telescope diameter in m
"time": t in s
"log_variance": $\log \sigma^2$
"throughput": f
"log_resolution": $\log R$
"wave_low": $\lambda_{\rm min}$
"wave_high": $\lambda_{\rm max}$

1.1 References

Morley, C.~V., Kreidberg, L., Rustamkulov, Z., Robinson, T., & Fortney, J.~J. 2017, ApJ, 850, 121