Observation Planning Part 2 - Sensitivity Calculation

1. Where is my target?
2. When can I observe my target?
3. How do I know when I've found my target (make a finder chart)?
4. **How long do I need to observe?**
5. How will I calibrate my data?

In Observation Planning part 1, we covered parts 1,2,3 above. Now we'll cover the 4th part, calculating observation times.

First, a brief review: For each of the following questions, give a brief explanation of what tools you use to answer the first three questions:

(1) Where is my target?

Your answer here

(2) When can I observe my target?

Your answer here

(3) How do I know when I've found my target?

Your answer here

How long do I need to observe my target?

We know approximately how long we *can* observe each target, but how long do we *need* per source? To answer this question, we need to do some sensitivity calculations.

First, we determine how bright our target is. You can only obtain this information for known sources, but generally, we can set a reasonable limit based off of physical considerations.

Our observations will be performed in the visual, meaning we are interested in the source brightness in the R (red) and V (visual) filters.

We can use SIMBAD to look up the typical brightness of the target source in magnitudes. You can go on the SIMBAD website (http://simbad.u-strasbg.fr/simbad/) to see how this works, but astroquery can also do it, so we're going to use astroquery here. The astroquery SIMBAD module tells you what "fields" (data column names) are available to query on for each source.

In [2]:
```
from astroquery.simbad import Simbad
import numpy as np
```

Recall from the previous workbook that we can use SIMBAD to retrieve information about stars, but the information included does not, by default, include the flux (or magnitude) of the source.

In [3]:
```
pcyg_simbad = Simbad.query_object('P Cygni')
pcyg_simbad
```

Out[3]: *Table length=1*

MAIN_ID	RA	DEC	RA_PREC	DEC_PREC	COO_ERR_MAJA	COO_ERR_MINA	COO_ERR_A
	"h:m:s"	"d:m:s"			mas	mas	
object	str13	str13	int16	int16	float32	float32	
* P Cyg	20 17 47.2020	+38 01 58.552	11	11	0.120	0.130	

In [4]:
```
Simbad.list_votable_fields()
```

```
dim_bibcode
dim_incl
dim_majaxis
dim_minaxis
dim_qual

dim_wavelength
dimensions
distance
distance_result
einstein
fe_h
flux(filtername)
flux_bibcode(filtername)
flux_error(filtername)
flux_name(filtername)
flux_qual(filtername)
flux_system(filtername)
flux_unit(filtername)
```

We add the flux fields for the R and V filters, then we can see the two new columns added on the right:

In [5]:
```
Simbad.reset_votable_fields()
Simbad.add_votable_fields('flux(R)', 'flux(V)')
pcyg_simbad = Simbad.query_object('P Cygni')
pcyg_simbad
```

Out[5]:

EC	COO_ERR_MAJA	COO_ERR_MINA	COO_ERR_ANGLE	COO_QUAL	COO_WAVELENGTH	COC
	mas	mas	deg			
:16	float32	float32	int16	str1	str1	
11	0.120	0.130	90	A	O	2007A&A

P Cygni is about 4.3 magnitudes in R. What does that mean?

Magnitudes are a weird system with relative definitions. The definition of magnitude is:

$$m = -2.5 \log(F/F_0)$$

m is the observed (apparent) magnitude, F is the source flux (in units of Jansky, ergs/cm^2/angstrom, W/m^2/angstrom, or erg/cm^2/Hz, usually), and F_0 is the flux at magnitude zero.

F_0 is a definition!

It was originally referenced to Vega, but more accurate observations have revealed that Vega is itself variable, so now the absolute standard is set by physical units.

So, we have to look up F_0 or determine it ourselves.

Note that each filter will have its own zero-point. For example, for the visible V band, we say:

$$m_V = -2.5 \log(F_V/F_{0,V})$$

or for red:

$$m_R = -2.5 \log(F_R/F_{0,R})$$

In the red filter, the zero-point is approximately 2500 Jy. Therefore, we can compute the flux from P Cygni:

$$m_{PCyg} = 4.3 = -2.5 \log(F_{PCyg}/2500\text{Jy})$$

Solve for F_{PCyg}:

$$F_{PCyg} = 10^{(-4.3/2.5)} \times 2500\text{Jy}$$

In [6]:
```
from astropy import units as u
```

In [7]:
```
F_PCyg = 10**(-4.3/2.5) * 2500*u.Jy
F_PCyg
```

Out[7]: $47.636518\ \mathrm{Jy}$

This tells us the amount of energy received per area per unit frequency at the telescope. Next, we need to compute the number of photons we'll collect.

1 Jansky is 10^{-23} ergs s^{-1} cm^{-2} Hz^{-1} (https://en.wikipedia.org/wiki/Jansky (https://en.wikipedia.org/wiki/Jansky))

In [8]:
```
(1*u.Jy).to(u.erg/u.s/u.cm**2/u.Hz)
```

Out[8]: $1 \times 10^{-23}\ \frac{\mathrm{erg}}{\mathrm{Hz\ s\ cm^2}}$

We need to describe our instrument now. Our telescope is a 14" diameter telescope, so its area is (pi * (radius)^2). An inch is 2.54 centimeters. So, our collecting area is $A_{CTO14"} =$

In [9]:
```
# student answer here
A_CTO = np.pi*((7*(2.54))*(u.cm))**2
A_CTO
```

Out[9]: $993.14666\ \mathrm{cm^2}$

This should be 993.14666 cm^2

To determine how many photons we will receive, we need to know which wavelengths of photons are hitting the detector, and which ones are energetic enough to produce electrons through the photoelectric effect.

We do this by putting a filter in front of the detector, which prevents longer- or shorter-wavelength (lower- or higher-energy) photons from getting through.

We will approximate that our red filter is ~1000 angstroms wide (in reality, there is a wavelength-dependent filter function we could use (http://voservices.net/filter/), but we will take the simple approach for now). The width of a filter is relative to the wavelength, and the red filter is centered at about 6500 angstroms.

To convert a central wavelength from angstroms to Hertz, we use the definition $\nu\lambda = c$, i.e., the frequency times the wavelength is the speed of light.

For example, 100 microns (μm) is at 3 THz. Astropy's `constants.c` is the speed of light:

In [10]:
```
from astropy import constants
wavelength = 100*u.um
frequency = constants.c  / wavelength
frequency
```

Out[10]: $2997924.6 \ \frac{\mathrm{m}}{\mathrm{s}\ \mu\mathrm{m}}$

Note the units on that are something funky, $m\mu^{-1}s^{-1}$. You can use astropy's `.to` to convert to a specific unit, or `.decompose` to reduce to a minimal expressible unit:

In [11]:
```
print("Decomposed frequency: ", frequency.decompose())
print("Frequency in THz: ", frequency.to(u.THz))
```

```
Decomposed frequency:  2997924580000.0 1 / s
Frequency in THz:  2.99792458 THz
```

So, returning to our red filter, for a wavelength $\lambda = 6500$ angstroms, solve for ν in Hz or THz:

In [12]:
```
# student answer here
lam = 6500*u.angstrom
nu = constants.c / lam.decompose()
nu.to(u.Hz)
```

Out[12]: 4.6121917×10^{14} Hz

Should be 4.6121917×10^{14} Hz

The filter width is $\Delta\lambda/\lambda$. We have defined the width $\Delta\lambda = 1000$ angstroms, and the central wavelength is $\lambda = 6500$ angstroms. The relative width is the same in both units, i.e.:

$$\frac{\Delta\lambda}{\lambda} = \frac{\Delta\nu}{\nu}$$

You solved for ν above, so now determine $\Delta\nu$ (again, in Hz or THz)

In [13]:
```
# student answer here
dnu = nu*((1000)/(6500))
dnu.to(u.Hz)
```

Out[13]: 7.0956795×10^{13} Hz

Your answer should be 7.0956795×10^{13}Hz

We can convert our flux in Janskys to 10^{-23} ergs s^{-1} cm^{-2} Hz^{-1}, then multiply by the bandwidth in Hz and the collecting area in cm^2 to obtain the energy we will receive on the detector per second:

$$E_{rec} = F_{PCyg} * A_{CTO14"} * \Delta\nu$$

Compute E_{rec} in units of erg s^{-1} (`u.erg/u.s`):

In [14]:
```
# you should have these three variables defined already
dnu, F_PCyg, A_CTO
```

Out[14]:
```
(<Quantity 7.09567948e+13 1 / s>,
 <Quantity 47.63651795 Jy>,
 <Quantity 993.14665903 cm2>)
```

In [15]:
```
(1*u.Jy).to(u.erg/u.s/u.cm**2/u.Hz)
```

Out[15]: $1 \times 10^{-23} \ \frac{\mathrm{erg}}{\mathrm{Hz\ s\ cm^2}}$

In [16]:
```
# student answer here (in erg/s)
dnu = dnu.to(u.Hz)
F_PCyg = F_PCyg.to(u.erg/u.s/u.cm**2/u.Hz)
print(dnu, A_CTO,F_PCyg)

E_rec = (F_PCyg)*A_CTO*dnu
E_rec.to(u.erg/u.s)
```

```
70956794792899.39 Hz 993.1466590310957 cm2 4.763651794908118e-22 erg / (cm2 Hz
s)
```

Out[16]: $3.3569694 \times 10^{-5} \ \frac{\mathrm{erg}}{\mathrm{s}}$

Should be: 3.3569694×10^{-5} ergs / s

The CCD detector responds to individual photons, so we need to count how many photons are coming in. The average wavelegnth of a photon is 6500 angstroms (approximately). We can compute the energy per photon using

$$E_{phot} = h\nu$$

where h is Planck's constant (https://en.wikipedia.org/wiki/Planck_constant). Using the above information, compute the mean energy per photon.

(recall that frequency and wavelength are related as $c = \nu/\lambda$)

In [17]:
```
# student answer here (compute the anwer in both eV and erg)
E_phot = constants.h*(nu)

print(E_phot.to(u.eV))
print(E_phot.to(u.erg))
```

```
1.9074492066646191 eV
3.0560705494598897e-12 erg
```

You should obtain a number near 2 eV.

Given our energy received from the star E_{rec} [erg/s] and the energy per photon E_{phot} [erg], how

many photons are we receiving per second?

In [18]:
```
# student answer here (units (1/u.s))

phot_per_s = E_rec/E_phot.to(u.erg)
phot_per_s
```

Out[18]: $10984594 \ \frac{1}{\mathrm{s}}$

Answer should be: 10984594 s^{-1}

CCD detectors saturate at approximately $2^{16} = 65536$ counts (though ours saturate at closer to 30,000). Is the photon count greater than or less than this number?

Greater

If the above is greater than the saturation count, how many pixels do we need to spread the photons over to stay below saturation in 1 second?

In [19]:
```
# student answer
## counts = 10984594 for 1 s exposure
## 30000 / 10984594
npix = 10984594 / 30000
npix
```

Out[19]:
```
366.15313333333336
```

The CCDs at CTO are ST-402 cameras. They have 765x510 pixels over 6.9x4.3mm.

How big are the individual pixels, in microns? (should be about 9x9 microns)

In [20]:
```
# student answer: compute the pixel size in both dimensions
## total_pix = 765 len 510 width
## total_pix_area = 6.9 mm len by 4.3 mm width
pix_len = 6.9*(u.mm)
pix_wid = 4.3*(u.mm)
pix_size_len = pix_len/765
print(pix_size_len.to(u.um))

pix_size_wid = pix_wid/510
print(pix_size_wid.to(u.um))
```

```
9.019607843137257 um
8.431372549019608 um
```

By observing a pair of stars with known separation on the sky, we can determine what the *pixel scale* of our CCD is.

However, for the purpose of this exercise, we will simply assume that the pixel scale is ≈ 0.5 " per pixel.

'

Given the typical seeing at CTO (about 2"), over what radius circle does our star need to be projected to avoid saturating any pixels?

We calculate this in two steps: First, we know that the *area* we need to project onto must be ~168 pixels. What *radius* circle has an area of 168 pixels?

In [21]:
```
# student answer
## A = pi r^2   r = sqrt(A/pi)
radius_pixels = (168/np.pi)**(1/2)*(u.pixel)
radius_pixels
```

Out[21]: $7.3127328 \mathrm{pix}$

Should be 7.3 pixels

What does that size in pixels correspond to in arcseconds?

In [22]:
```
pixel_scale = 0.5*u.arcsec/u.pixel
```

In [23]:
```
radius_pixels * pixel_scale
```

Out[23]: $3.6563664 \mathrm{''}$

Is the radius we need to smear the source over, i.e., the radius in arcseconds we just calculated above, bigger or smaller than the expected source size (the "point spread function")?

.5 arc sec per pixel

(0.5)(7.3) = 3.65

The expected source size is the same

Given that answer, can we observe this target without saturating the CCD? Justify your answer.

We should be able to observe it because we can techically spread it all out

In [24]:
```
(0.5)*(7.3)
```

Out[24]:
```
3.65
```

Integration Times

In the above, we were simply determining whether the source would be above or below the saturation level of our CCDs. Let's now determine how long we need to integrate to achieve a target signal to noise ratio.

We decide our integration time based on a target signal-to-noise ratio (SNR). If our target SNR is

100, how long do we need to integrate?

To a first approximation, we can assume that the signal-to-noise is limited by counting statistics. That means that the signal S rises as the count rate $\dot{p}$ times the time, $S = \dot{p}t$. The noise N is, in the Poisson noise limit, the square root of the signal (in photons), i.e., $N = \sqrt{(S)}$. The SNR is therefore

$$S/N = \frac{\dot{p}t}{\sqrt{\dot{p}t}} = \sqrt{\dot{p}t}$$

Warning / Note

When we say that the uncertainty is the square root of the signal when counting photons, we have a bit of a units problem!

$$\sigma = \sqrt{\lambda}$$

The number of photons counted has units: number of photons. The *uncertainty* on that number has the same units, despite the fact that, naively, you would look at the above equation and say that the uncertainty has units of $\sqrt{counts}$.

You can't rely on the astropy unit system (or any automated unit system) to handle this equivalency; you need to take care to establish the correct units when dealing with uncertainties on counts.

We calculated the count rate above (`phot_per_s`), so what integration time t do we need for $SNR = 100$?

In [25]:
```
# student answer (in seconds)
## t = 100^2 / pdot
integration_time = (100**2)/(phot_per_s)
integration_time.to(u.ms)
```

Out[25]: $0.91036592\ \mathrm{ms}$

You should note that that is a pretty short time! (about 0.9 milliseconds)

What if we account for read noise? Let's assume that we are averaging over a 2 arcsecond Gaussian PSF. First, how many pixels is that?

The area of a Gaussian isn't exactly that of a circle. $A_{2D\mathrm{Gaussian}} = 2\pi\sigma^2$

In [26]:
```
psf_area = 2 * np.pi * (2*u.arcsec)**2
psf_area
```

Out[26]: $25.132741\ \mathrm{arcsec}^2$

In [27]:
```
psf_area_pixels = psf_area * pixel_scale**-2
psf_area_pixels
```

Out[27]: $100.53096\ \mathrm{pix}^2$

Given that our readnoise is 10 counts per pixel, and we're interested in the *sum* over these pixels, what is the expected contribution to the noise on the sum from these pixels?

In [28]:
```
# student answer
readnoise_per_pix = 10*u.adu/u.pix
readnoise_sum = (psf_area_pixels*readnoise_per_pix**2)**(1/2)
readnoise_sum
```

Out[28]: $100.26513\ \mathrm{adu}$

By coincidence, that's about 100.265 ADU

Given our calculation of the SNR and count noise above, what's our expected *total* noise in the measurement over the PSF? (recall: the signal will be $t \times \dot{p}$, and we have calculated both of those quantities above)

In [34]:
```
# student answer (note: this value should be in the same units as the read noise
## p dot = phot_per_s
## t = integration_time
## Signal S = t(p dot)
photon_noise = (phot_per_s*integration_time)**(1/2)*(u.adu)
photon_noise
```

Out[34]: $100\ \mathrm{adu}$

In [39]:
```
# student answer
total_noise = (photon_noise**2 + (readnoise_sum)**(2))**(1/2)
total_noise
```

Out[39]: $141.60896\ \mathrm{adu}$

Should be about sqrt(2) * 100

Given that our noise is higher than we originally thought, we will not achieve our target signal to noise ratio! What will we achieve instead?

In [112]:
```
# student answer
signal = (total_noise)**2
snr = signal / total_noise
snr
```

Out[112]: $141.60896\ \mathrm{adu}$

We can write down a different equation for the signal-to-noise ratio now that we know how much

readnoise to expect for a stellar observation.

$$S/N = \frac{\dot{p}t}{\sqrt{\dot{p}t + \sigma_{rn}^2}} \approx \frac{\dot{p}t}{\sqrt{\dot{p}t + 100}}$$

This equation is no longer trivially solvable, but it can be rearranged nicely enough into a quadratic equation:

$$SNR^2\ (\dot{p}t + 100) = (\dot{p}t)^2$$

if we substitute $S = \dot{p}t$, it looks a little simpler

$$S^2 - S \cdot SNR^2 - 100SNR^2 = 0$$

then, we can use the quadratic formula to solve for the needed signal:

$$S = SNR^2 \pm \frac{\sqrt{SNR^4 + 400SNR^2}}{2}$$

In [29]:
```
SNR = 100
target_signal = SNR**2 + (SNR**4 + 4*readnoise_sum.value*SNR**2)**0.5 / 2
target_signal
```

Out[29]: 15099.279489285953

In [30]:
```
integration_time_including_readnoise = target_signal / phot_per_s
integration_time_including_readnoise
```

Out[30]: 0.0013745869 s

In [31]:
```
# how much longer is that?
integration_time_including_readnoise / integration_time
```

Out[31]: 1.5099279

So, accounting for readnoise, to get to the same SNR, we need to integrate for about 50% longer.

Note that all of these calculations make a few assumptions that are generally not correct:

1. We assumed the V-filter has a uniform transmission of 1 across its band. In reality, the V-filter transmits closer to ~75% of the total band, so the total photon rate (in any given filter) is less than what we calculated above.
2. We assumed that the CCD has perfect *quantum efficiency*. The best CCDs have close to unity (one) efficiency, but not exactly. Ours are probably in the 60-80% range.
3. We assumed no flux losses to the atmoshpere (absorption or scattering) or the telescope optics. We know from class that the atmosphere absorbs anywhere from 10-30% of the incident flux if we point straight up on a clear night.

Together these effects amount to a substantial loss of light. Let's estimate how much:

In [32]:
```
filter_efficiency = 0.75
quantum_efficiency = 0.7
atmosphere_loss = 0.2
received_fraction = filter_efficiency * quantum_efficiency * (1-atmosphere_loss)
received_fraction
```

Out[32]:
```
0.41999999999999993
```

We're getting less than 50% of the light recorded on our CCD. That means we're liable to underestimate the brightness of our targets by a lot! Next time, we'll go over how we account for that loss by *calibrating*.

Do it yourself for a more reasonable target.

We noted above that P Cygni is really hard to observe, since it will saturate our detector.

Now, let's repeat the above exercises for a different target: `TYC 2788-108-1`

Answer the following questions (using the code you learned above):

1. How bright is TYC 2788-108-1 in the Johnson V-band (http://voservices.net/filter/filterdtls.aspx?filterid=60)? Give your answer in magnitudes and in Janskys, assuming the zero-magnitude flux is 3500 Jy

In [45]:
```
TYC = Simbad.query_object('TYC 2788-108-1')
TYC
```

Out[45]:

EC	COO_ERR_MAJA	COO_ERR_MINA	COO_ERR_ANGLE	COO_QUAL	COO_WAVELENGTH	COC
	mas	mas	deg			
t16	float32	float32	int16	str1	str1	
14	0.035	0.048	90	A	O	2018yCa

In [47]:
```
Simbad.reset_votable_fields()
Simbad.add_votable_fields('flux(R)', 'flux(V)')
TYC = Simbad.query_object('TYC 2788-108-1')
TYC
```

Out[47]:

EC	COO_ERR_MAJA	COO_ERR_MINA	COO_ERR_ANGLE	COO_QUAL	COO_WAVELENGTH	COC
	mas	mas	deg			
t16	float32	float32	int16	str1	str1	
14	0.035	0.048	90	A	O	2018yCa

In [98]:
```
F_TYC = 10**(-10.65/2.5) * 3500*u.Jy
F_TYC
```

Out[98]: $0.19233931\ \mathrm{Jy}$

In [99]:
```
F_TYC.to(u.erg/u.s/u.cm**2/u.Hz)
```

Out[99]: $1.9233931 \times 10^{-24}\ \frac{\mathrm{erg}}{\mathrm{Hz\ s\ cm^2}}$

2. Using the same telescope as above, what is our expected received flux in ergs/second and in photons per second? Assume the V-filter band width is 870 Angstroms and the band center is 5504 Angstroms.

In [127]:
```
## v filter band width 870 angstroms
## band canter 5504 angstroms
F_TYC = F_TYC.to(u.erg/u.s/u.cm**2/u.Hz)

lamT = 5504*u.angstrom
nuT = constants.c / lamT.decompose()
nuT.to(u.Hz)

dnuT = nuT*((870)/(5504))
#dnuT = dnuT.to(u.Hz)

E_recT = (F_TYC)*A_CTO*(dnuT)
print(E_recT.to(u.erg/u.s))

E_photT = constants.h*(nuT)

Tphot_per_s = E_recT/E_photT
print(Tphot_per_s.to(u.Hz))
```

```
1.6446160921481125e-07 erg / s
45568.65689848281 Hz
```

In [126]:
```
#dnuT = (constants.c / (5504*u.angstrom).to(u.m)).to(u.Hz)
#print(nuT)
#
#dnuT = nuT*((870*u.angstrom)/(5504*u.angstrom))
#print(dnuT)
#
#E_recT = ((F_TYC)*A_CTO*(dnuT)).to(u.erg/u.s)
#print(E_recT)
#
#E_photT = constants.h*(nuT)
#
#Tphot_per_s = E_recT/E_photT
#print(Tphot_per_s.to(u.Hz))
```

```
544681064680232.5 Hz
86096025848801.28 Hz
1.6446160921481125e-07 erg / s
45568.65689848281 Hz
```

In [115]:
```
E_photT.to(u.eV), E_photT.to(u.erg)
```

Out[115]:
```
(<Quantity 2.25261988 eV>, <Quantity 3.60909494e-12 erg>)
```

In [132]:
```
Tphot_per_s = (E_recT / E_photT).to(1/u.s)
Tphot_per_s
```

Out[132]: $45568.657 \ \frac{1}{\mathrm{s}}$

3. How many photons per second per pixel will the star produce if the point spread function, the area over which the star's light is spread, is the same as above ($\sigma_{Gaussian} = 2$ ")? How long can we integrate without saturating (assuming we saturate at 30,000 counts)?

In [137]:
```
#npixT = 45568.65689848281 / 30000
#print(npixT)
#
Tspread = Tphot_per_s / np.sqrt(psf_area_pixels)
print(Tspread)

integration_time_including_readnoiseT = target_signal / Tspread
print(integration_time_including_readnoiseT)

integration_time1T = (100**2)/(Tspread)
print(integration_time1T)
```

```
4544.815974477803 1 / (pix s)
3.322308224156612 pix s
2.2003091117785014 pix s
```

4. Again following the same assumptions (the same spread among pixels, same readnoise), how long do we need to integrate to get to a signal-to-noise ratio of 100? Account for both photon statistics and readnoise

In [143]:
```
## same spread along pixels same readmoise
## how long to integrate for SNR = 100

integration_timeT = (100**2)/(Tphot_per_s)
print(integration_timeT.to(u.ms))

photon_noiseT = (Tphot_per_s*integration_timeT)**(1/2)*(u.adu)
print(photon_noiseT)

total_noiseT = (photon_noiseT**2 + (readnoise_sum)**(2))**(1/2)
print(total_noiseT)

signalT = (total_noiseT)**2
snrT = signal / total_noiseT
print(snrT)

integration_time_including_readnoiseT = target_signal / Tphot_per_s
print(integration_time_including_readnoiseT)
```

```
219.4490836602416 ms
100.0 adu
141.6089562544945 adu
141.6089562544945 adu
0.3313523047853683 s
```

In [ ]:
```
```