Supernova Bootcamp

Version 1.0.2

Lauren Aldoroty

June 28, 2023

Contents

1 Introduction

Compiling all the things I've learned so it's easier for future folks. This manual will attempt to cover more technical, data-analysis driven perspectives on working with SNe, particularly SNe Ia. In the Software section (Section 7), at the end of each subsection, I'll list some lesser-known functions and tricks that I've found useful. These will be indicated by section titles `in font like this`, e.g., `helio_to_cmb()` (Section 7.2.3).

This manual is perpetually incomplete. Some sections may be partially written, and some aren't written at all because I've put the header there so I remember to write it in the future. I'm happy to receive comments and suggestions. E-mail me: laurenaldoroty [at] gmail [dot] com.

2 Reddening and Extinction

Reddening and extinction are *nearly* the same thing, but not quite. They're both caused by light scattering off dust, but *extinction* describes the dimming that results, and *reddening* describes the preferential scattering of blue light (i.e., more shorter wavelength light gets scattered away from us than longer wavelengths, so we see objects as redder than they actually are.) This is important because it's a pretty large source of systematic uncertainty in supernova studies! Dust doesn't emit light, it just takes it away, so we can't "see" it directly. *HOW* are we supposed to correct for an effect we know so little about?

2.1 Background

Extinction, in general, is described by the equation:

$$A_V = -2.5 \log \frac{F_V}{F_{V,0}}, \tag{1}$$

where the subscript V indicates the V-band photometric bandpass, F_V is observed flux in the V-band, and $F_{V,0}$ is the intrinsic flux in the V-band.

Reddening is described by:

$$E(B-V) = (B-V)_{observed} - (B-V)_{intrinsic}, \tag{2}$$

where $(B-V)_{observed}$ is the observed $B-V$ photometric color, and $(B-V)_{intrinsic}$ is the intrinsic photometric color. Note that because magnitudes are logarithmic, this equation is a ratio, too! We can link extinction and reddening with an equation, as well:

$$E(B-V) = A_B - A_V, \tag{3}$$

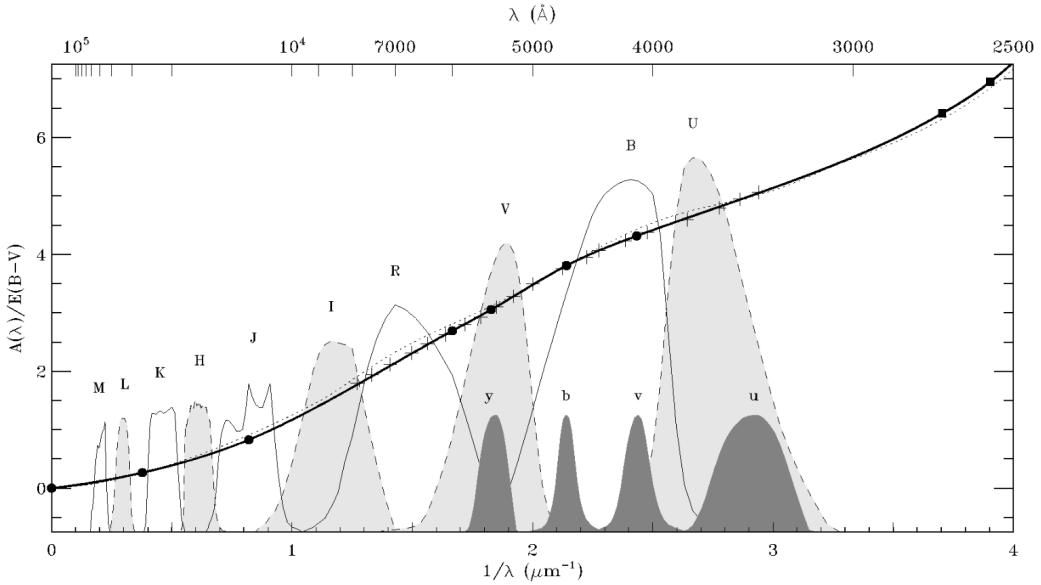

Figure 1: R_X (X for an arbitrary filter) for Johnson photometric filters (transmission functions shown scaled arbitrarily for reference). Solid points are data, the solid line is a cubic spline interpolation (*not* a fit). Note that extinction has a more significant effect at shorter wavelengths. Figure from [1].

where A_B is photometric extinction in the B-band and A_V is photometric extinction in the V-band. We can also link them like this:

$$A_V = R_V E(B-V), \tag{4}$$

where R_V is a constant called the "total-to-selective extinction ratio". Basically, it's a constant that tells you how much light you lose in a particular photometric band.

2.2 Correcting for Dust

There are a few packages you can use to quickly un-dust your data. First, `dust_extinction` has a bunch of dust extinction curves and allows you to redden or deredden the spectrum by multiplying or dividing by the dust model's `extinguish` attribute, respectively. It also takes care of units for you, so you'll need `astropy`'s `units` module. You have to provide it with $E(B-V)$ or A_V as well. I like to get that with the SNooPy `get_dust_RADEC()` function, which retrieves the Milky Way $E(B-V)$ from IRSA based on RA and Dec (Section 7.3.1). Here's some pseudocode showing how I deredden the flux for a spectrum:

```
from astropy import units as u
from dust_extinction.parameter_averages import F04
```

```
from snpy.utils.IRSA_dust_getval import get_dust_RADEC

dustmodel = F04(Rv=3.1)
mwreddening = get_dust_RADEC(ra, dec, calibration='SF11')[0][0]
dereddened_flux = ((flux*u.erg/(u.s*u.AA*u.cm**2))/
    dustmodel.extinguish(wave*u.AA, Ebv=mwreddening)).value
```

3 Hubble's Law and SN Ia Cosmology

3.1 What the heck is a rest frame?

Let's say you have some data, but it's from a supernova with $z = 0.1$. Because of the relativistic Doppler effect, the wavelengths you *observe* are different from the "actual" wavelengths, i.e., rest frame wavelengths. It's called *red*shift for a reason—observed wavelengths will be longer, or redder, than the rest frame wavelengths. This effect will also affect observed flux. Your observed flux is smaller than your rest frame flux because longer wavelengths are less energetic than shorter ones. Ignoring ejecta velocity (see Section 5.3.2), this is why a spectral line won't appear in its "expected", or rest frame, position. Usually, you want to put your spectra in the rest frame before doing anything with them because it's hard to compare SNe at different redshifts. See Section 5.1 for details on correcting SN spectra to the rest frame.

3.2 z_{helio} and z_{CMB}

You'll encounter two kinds of redshift—heliocentric redshift (z_{helio}) and CMB (Cosmic Microwave Background) redshift (z_{CMB}). *These are different, and are used for different purposes!* z_{CMB} is the redshift caused by the Universe's expansion *only*, i.e., where the reference frame is the CMB frame. z_{helio} is the redshift with *only* the Earth's rotation and orbit removed. There are still effects from other motion, like Galactic rotation and the Galaxy moving around with respect to other objects, as well as z_{CMB}. "Heliocentric" $\equiv$ "Sun at center", so "Sun at center" is the rest frame. It's how things are moving relative to the Milky Way. Note that z_{helio}, while often reported as an object's redshift, is *not* the observed redshfit. The observed redshift does not have the Earth's rotation and orbit removed. In other words, z_{helio} contains information about a bunch of things moving relative to each other, including motion relative to the CMB, while z_{CMB} contains information about *only* motion relative to the CMB.

For cosmology, you'll use z_{CMB} (e.e., when making a Hubble diagram or determining other cosmological parameters). If you're dealing with data that you need to correct to the rest frame, you probably want to use z_{helio}.

3.3 Peculiar velocity and the Hubble flow

If you're reading this section, you may have heard the term "smooth Hubble flow" thrown around a few times. But what *is* this thing?

for one, not all of the components of a galaxy's velocity are moving away in accordance with the Universe's expansion. This is called the *Hubble flow*—if an object is *only* moving due to expansion (mostly), then it's in the Hubble flow. Any velocity deviation from the Hubble flow is called *peculiar velocity*. For example, if a galaxy is under gravitational influence from another galaxy, the components of its motion due to that interaction is part of its peculiar velocity. However, this is hard to quantify, so it's usually assumed that peculiar velocity is around 300 km/s. You'll notice that in plots that show peculiar velocity error, like in the lower panel of Figure 3, the error due to peculiar velocity gets smaller as an object is farther away.

Peculiar velocity error is calculated as:

$$\sigma_{v_{pec}} = \left(\frac{5}{\ln 10}\right)\left(\frac{v_{pec}}{cz}\right) \tag{5}$$

So, it's a function of z! This makes sense—the farther away something is, the faster it's moving in the Hubble flow, so the peculiar velocity is a relatively small portion of its net velocity vector.

3.3.1 Correcting for peculiar velocity

It's important to correct your z for peculiar velocity—it influences your results [2]! You can get your peculiar velocity values using this Python package, called `pvhub`. The code is originally written by Erik Peterson for [2], but I modified it so it is an installable Python package. This paper has a really good overview of the different components of z, so check it out. For the purposes of this section, we care about correcting z_{CMB} for v_{pec}. So, I'll parrot what's in that paper for a sentence or two. We go from v_{pec} to z_{pec} like this:

$$z_{pec} = \frac{v_{pec}}{c}, \tag{6}$$

and correct z_{CMB} like this:

$$z_{cosmo} = \frac{1 + z_{CMB}}{1 + z_{pec}} - 1. \tag{7}$$

3.4 How to make a Hubble diagram

Firstly, what is a Hubble diagram? It's a plot of recession velocity (y-axis) vs. distance (x-axis) (Figure 2). Turns out these things are positively correlated. In plain English, the farther away a galaxy is, the faster it's hurtling away from

us (and everything else—the Universe is homogeneous and isotropic). This means the Universe is expanding[1]!

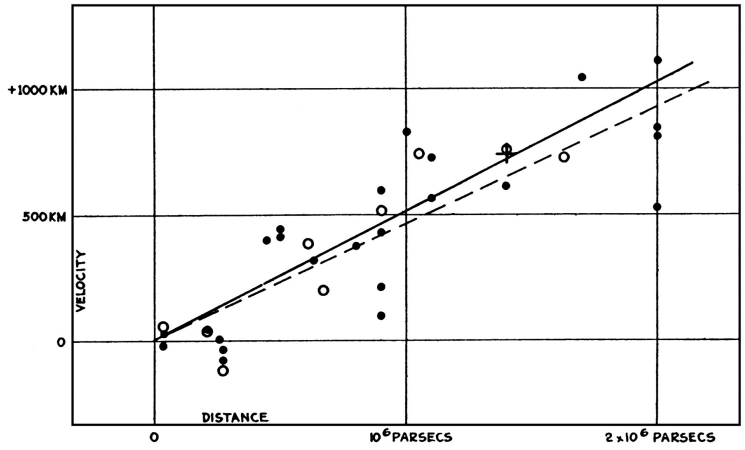

Figure 2: The OG Hubble diagram. Note that the velocity units are km—*this is a mistake*. It should probably be km/s. Even Edwin Hubble made an error on his most famous published figure. You're doing just fine.

However, no one really uses linear distance units, like parsecs, or velocities like parsecs per second in Hubble diagrams in the literature any more. You still might see these, though. Units you may see to represent recession velocity on the y-axis include:

- km/s
- The distance modulus μ (apparent magnitude minus absolute magnitude, $m - M$.)

Units you may see to represent distance on the x-axis include:

- Mpc
- z
- $\log(z)$ or $\log(cz)$

You'll most commonly see the distance modulus. It's the difference between apparent magnitude, m, and absolute magnitude, M. So, if the difference between these quantities is large, then an object is further away. If the difference is small, it's closer. You can think of M as the intrinsic brightness of an object. m is the brightness that you observe, corrected for effects like dust extinction (and because we're supernova people, also stuff like the luminosity-decline rate relation).

[1]It's expanding at an accelerating rate, but this statement and a very crude Hubble diagram alone do not imply accelerated expansion

So, there are many ways you can actually make a Hubble diagram because there are a zillion ways to standardize SNe Ia. I'll discuss χ^2 minimization i Section 9.4.

3.5 Hubble residual

The Hubble residual is the difference between the observed distance modulus of an SN and the expected distance modulus, obtained by fitting a cosmology to the data. This is important because this quantifies the uncertainty in our cosmological parameters. So, a lot of work is centered around minimizing the Hubble residual.

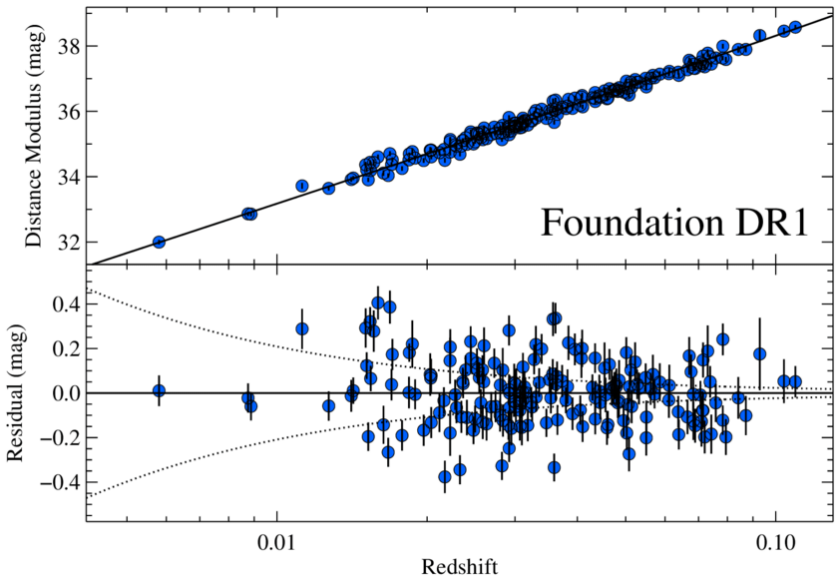

Figure 3: A Hubble diagram (top) made with SNe Ia. The Hubble residual is plotted in the lower panel. The curved dotted lines in the lower panel mark error due to peculiar velocity. Figure from [3].

The curved dotted lines in Figure 3 are estimated error due to peculiar velocity (Section 3.3, Equation 5), usually assumed to be around 300 km s^{-1}. While you'll plot those curved dotted lines as peculiar velocity error alone, you should add the error due to peculiar velocity in quadrature (i.e. sum the errors like $\sigma = \sqrt{a^2 + b^2}$) to the error bars on your data.

3.5.1 Troubleshooting

My Hubble residual shows a slight trend: this could be becuase you need to correct z_{CMB} for the effects of peculiar velocity. See Equation 7.

My scatter is just... enormous: if you used least squares minimization, check that (1) you typed your χ^2 expression correctly, and (2) that you're

inputting the correct arrays (e.g., if you accidentally put in m_B where σ_{m_B} should be).

3.6 Important papers for supernova cosmology

The Use of Supernovae Light Curves for Testing the Expansion Hypothesis and Other Cosmological Relations, B. Rust's PhD thesis, 1974 [4]. The author shows that SNe Ia can be used to test the expanding Universe hypothesis.

Light curves, color curves, and expansion velocity of type I supernovae as functions of the rate of brightness decline, I. Pskovskii 1977 [5]. This is the first paper to quantify the rate of brightness decline. It was quantified as β, which paved the way for Δm_{15} much later [6].

The Absolute Magnitudes of Type Ia Supernovae, M. Phillips 1993 [6]. This paper shows that the absolute magnitudes of SNe Ia are correlated with the decline rate of the $B-$band light curve. This paper is why the luminosity-decline relation is sometimes called the "Phillips relation". This paper quantifies the brightness decline rate as Δm_{15}—the change in magnitude from the date of maximum brightness to 15 days later.

Dr. Paulina Lira's Masters thesis, 1995. This thesis notes a linear region in the color curve (color vs. time) of SNe Ia. This is suggested as a method to correct reddening.

The Reddening-Free Decline Rate Versus Luminosity Relationship for Type IA Supernovae, M. Phillips et al. 1999. [7]. This paper elaborates on Phillips 1993 [6]. It modifies the previously-described luminosity-decline relation by correcting for host galaxy extinction using the relation in Lira 1995.

Observational Evidence from Supernovae for an Accelerating Universe and a Cosmological Constant, Riess et al. 1998 [8]. This is the paper that won the Nobel Prize in 2011, awarded to Adam Riess, Saul Perlmutter, and Brian Schmidt. The authors used High-z Supernova Search Team data combined with some nearby SNe to show that the distances of the high z SNe were larger than expected, implying that the Universe is expanding at an accelerating rate. The Supernova Cosmology Project group found the same results at around the same time.

4 Photometry

Caveat about this section: I have only ever used the Vega magnitude system. There will be no discussion of AB magnitudes.

4.1 Converting from flux to magnitude

Here, we just use the definition of magnitudes:

$$m = -2.5 \log_{10} \left(\frac{F_{obs}}{F_{ref}} \right), \tag{8}$$

where F_{obs} is your observed flux and F_{ref} is your reference flux in the *same* photometric band. You need to make sure your units are consistent between F_{obs} and F_{ref}. *You need to understand your magnitude system before doing this.* I repeat, *YOU NEED TO UNDERSTAND YOUR MAGNITUDE SYSTEM BEFORE DOING ANYTHING!* Know your units. Know your initial units, know your end units. Know the units of your reference, know the units of your data.

Using equation 41, we can also convert flux error to magnitude error:

$$\sigma_m = \sqrt{\left(\frac{2.5}{\ln 10} \right)^2 \left(\frac{\sigma_{F_{obs}}}{F_{obs}} \right)^2} \tag{9}$$

4.2 Converting from magnitude to flux

Rearrange the definition of magnitudes:

$$F_{obs} = F_{ref} 10^{\frac{-m}{2.5}} \tag{10}$$

Then, if you want to convert the magnitude error to flux error, use equation 41 on this equation, and get:

$$\sigma_{F_{obs}} = \sqrt{\left(\sigma_m \frac{F_{ref} \ln(10)}{2.5} 10^{\frac{-m}{2.5}} \right)^2} \tag{11}$$

I'd like to issue a word of caution on converting from magnitude units back to flux units. If you convert a magnitude in a particular magnitude system, e.g. Vega, back to flux units, then the flux you've computed is the flux of the object *in the Vega system, not the physical flux of the source.* This first came up for me when using `sncosmo`, so if you're having some zeropoint troubles with that code and you converted from magnitudes to flux, this is likely the issue. Check out 7.2 for the procedure to fix this.

5 Spectra

5.1 Correcting to the rest frame

Correcting the observed wavelength for the relativistic Doppler effect is done by:

$$\lambda_{rest\ frame} = \frac{\lambda_{observed}}{1 + z_{helio}}. \tag{12}$$

Correcting the observed flux is more complicated and I've redacted this section for now. If you need to correct flux to the rest frame, see the next section and normalize to a chosen z.

5.2 Normalizing to a particular z

Sometimes, you want your spectrum flux as if it were at some redshift z_{ref} that you've chosen. Do not use the rest frame spectrum for this calculation. First thing's first, you need the luminosity distance to your object.

$$D_L = (1 + z_{helio})D_M, \tag{13}$$

where z_{helio} is the heliocentric ($\sim$observed) redshift for your object, and D_M is the comoving transverse distance. In Python, you can use `astropy.cosmology`'s function `comoving_transverse_distance`, which takes z_{CMB} as its input (so you'll also need your object's RA and dec), to calculate D_M. If we assume an FLRW universe, then our cosmological scale factor is $a(t) = 1/(1 + z)$. Then,

$$\frac{a(t_{ref})}{a(t_{actual})} = \Big(\frac{1 + z_{helio}}{1 + z_{ref}}\Big)\Big(\frac{D_{L,actual}}{D_{L,ref}}\Big)^2 \tag{14}$$

Finally, convert your flux by using this value:

$$F_{ref} = F_{actual}\Big(\frac{1 + z_{helio}}{1 + z_{ref}}\Big)\Big(\frac{D_{L,actual}}{D_{L,ref}}\Big)^2 = F_{actual}\Big(\frac{a(t_{ref})}{a(t_{actual})}\Big) \tag{15}$$

Using Equation 41, the error is:

$$\sigma_{F_{ref}} = \sigma_{F_{actual}}\frac{\partial F_{ref}}{\partial F_{actual}} = \sigma_{F_{actual}}\Big(\frac{1 + z_{helio}}{1 + z_{ref}}\Big)\Big(\frac{D_{L,actual}}{D_{L,ref}}\Big)^2 \tag{16}$$

5.3 Absorption lines

5.3.1 Pseudo-equivalent width

Pseudo-equivalent width (pEW) is a measure of the strength of an absorption line. It combines the depth and the width of the line into one measurement by integrating over the feature. There's a really good run-down of pEW in [9]. So, we're going to borrow from them. Figure 4 shows an absorption feature. You start with one like the one on the left (red). We want to measure the strength of the absorption feature, but the problem with SNe is that there are so many absorption and emission lines and there's doppler shift everywhere, so we don't really know where the continuum is. *But*, we need a way to fairly compare absorption features between different SNe, which means removing the continuum. So, we make a "pseudo-continuum" to approximate a small region of the actual continuum. Lines are nice, so all you do to make the pseudo-continuum is draw a line across the top of the absorption feature.

We remove the "continuum" by normalizing the absorption feature to the pseudo-continuum. You only do this for the one feature—for each absorption feature, you need a new pseudo-continuum. After you've normalized to the pseudo-continuum, you end up with the *right* panel of Figure 4 (green). Now, you can measure the area and fairly compare line strengths between SNe.

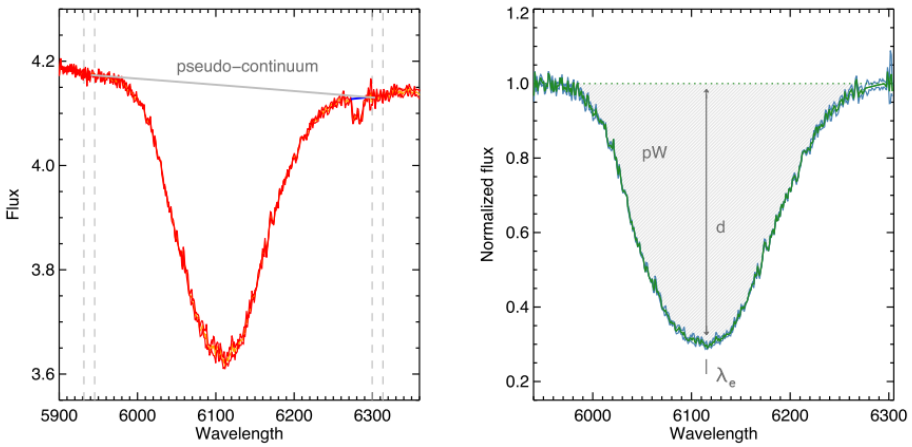

Figure 4: *Left*: Un-normalized absorption feature. *Right*: The same absorption feature normalized to the pseudo-continuum marked on the *left*. d is the feature depth and λ_e is the center wavelength. The shaded area is the pEW (pW in the figure). Figure from [9].

That's great and all, but how do we actually do all this in practice? There are a lot of ways. Let's break it down into steps and talk about 'em. This is *not* an exhaustive list, nor is it the only way to do this—these are merely suggestions. Let your imagination run free, spectral padawan.

1. **Finding the pseudo-continuum:** The challenging part of this step is avoiding noise. We don't want an unrealistic pseudo-continuum because we picked a point where there's a statistical spike or dip in the measurement. You can smooth the data with a moving average, or use the Savitzky-Golay filter from `scipy.signal.savgol_filter()`. Discussing the differences between these two is outside the scope of this manual, but you should always understand the techniques you're using!

2. **Normalizing to the pseudo-continuum:** Take your smoothed flux and divide by the line you drew. Here's some Python-inspired pseudo-code to help you out:

```
continuum = scipy.interpolate.interp1d(flux[start_point],
    flux[end_point])
normalized_flux = flux[start_point:end_point]
    /continuum(wavelength[start_point:end_point])
```

3. **Integrating over the absorption feature:** You can do this via summation directly over the normalized data or by fitting a Gaussian and integrating that.

Now... we weren't going to have this entire discussion without talking about error on pEW, of course. The error for this kind of "guesstimation" thing is hard to quantify precisely, so I recommend a bootstrapping method like the one in [9]. Instead of choosing fixed endpoints for your pseudo-continuum, draw randomly from some small region around your chosen endpoints N times to get N sets of different, but all reasonable, endpoints. Now, calculate the pEW as discussed above for each set of endpoints. Then, your final pEW measurement is the mean of all N measurements, with the standard deviation of all N measurements as your error on pEW.

5.3.2 Ejecta velocity

5.4 Synthetic photometry

If you're here, you're wondering how to make photometry from some spectra. In short, this is obtained by integrating under the spectrum in the region covered by a given filter.

The area under the spectrum, F, in a given filter X, is calculated by

$$F_X = \frac{1}{hc}\int_{\lambda_1}^{\lambda_2} \lambda f(\lambda) R_X(\lambda) d\lambda, \tag{17}$$

where $f(\lambda)$ is your spectrum, and $R_X(\lambda)$ is your filter. The λ and $1/hc$ terms are in there if we're using photon-counting detectors, and using an input spectrum that's in energy units. If you're using an energy-counting detector, you would drop this, i.e., your integrand is $f(\lambda)R_X(\lambda)d\lambda$.

However, we can't often use the continuous version of this function with our data. So, we discretize it:

$$F_X = \frac{1}{hc}\sum_{i=\lambda_1}^{\lambda_2} \lambda_i f(\lambda_i) R_X(\lambda_i)(\lambda_i - \lambda_{i-1}), \tag{18}$$

where $f(\lambda_i)$ is the energy flux at a particular wavelength λ_i, and $R_X(\lambda_i)$ is the response function for the filter at wavelength λ_i. We're using a discretized version of an integral, so $(\lambda_i - \lambda_{i-1})$ is the same as $d\lambda$. From now on, this chapter will be written as if we are using Equation 18.

Then, for a photon-counting detector with an input spectrum in units of energy, the variance of F (using Equation 41) is:

$$\sigma_F^2 = \frac{1}{(hc)^2}\sum_{i=\lambda_1}^{\lambda_2} \lambda_i^2 \sigma_{f(\lambda_i)}^2 R_X(\lambda_i)^2 (\lambda_i - \lambda_{i-1})^2 \tag{19}$$

For an energy-counting detector with an input spectrum in units of energy, the variance of F is:

$$\sigma_F^2 = \sum_{i=\lambda_1}^{\lambda_2} \sigma_{f(\lambda_i)}^2 R_X(\lambda_i)^2 (\lambda_i - \lambda_{i-1})^2 \tag{20}$$

Individual magnitudes are calculated by

$$m = -2.5 \log_{10}\left(\frac{F}{F_{ref}}\right) = -2.5 \log_{10}(F) + 2.5 \log_{10}(F_{ref}) = -2.5 \log_{10}(F) - m_{ref}. \tag{21}$$

Wait—what are F_{ref} and m_{ref}? Good question. Because magnitudes are inherently relative quantities, we need to use some reference object to convert flux into magnitudes. You can get reference objects from CALSPEC. Photometric systems are, unfortunately, out of the scope of this manual at this time. Let's say we're using the Vega system, with the star Vega (Alpha Lyrae) from now on.

This means we need to use Equation 18 on our reference object, Vega to calculate F_{ref}. You'll use the *same* response function as you use for your supernova spectrum. This means that m_{ref} is the... reference magnitude? What does THAT mean? If magnitudes are inherently relative quantities, do we then compare our reference object to something *else*? A valid concern, but no! This is called a *zero point*. For the Vega system, astronomers have defined the magnitude of the star Vega such that the $m_{Vega} = 0$ in all bands. So, if you're using the Vega system, m_{ref} is probably just 0 (unless some literature tells you otherwise, which is possible, so make sure you're familiar with the photometric system and filters you're using).

So anyway, the variance of m (for an energy-counting detector) is:

$$\sigma_m^2 = \frac{\partial m}{\partial f}^2 \sigma_f^2 = \frac{\partial m}{\partial F}^2 \Big(\sum_{i=\lambda_1}^{\lambda_2} \frac{\partial F}{\partial f(\lambda_i)} \sigma_{f(\lambda_i)} \Big)^2 = \Big(\frac{-2.5}{F \ln 10} \Big)^2 \sum_{i=\lambda_1}^{\lambda_2} \sigma_{f(\lambda_i)}^2 R_X(\lambda_i)^2 (\lambda_i - \lambda_{i-1})^2 \tag{22}$$

Note that the summation term on the right-hand side of the equation is the same as σ_F^2. You need to be careful about flux variance here, though: Equations 18, 19, and 20 are true, but when you are calculating magnitudes, keep in mind that you're calculating this based off a relative flux. So, your flux variance should also be relative. In other words, it should be the variance of F/F_{ref}. Using Equation 41 and treating F_{ref} as a constant, we get

$$\sigma_{F/F_{ref}}^2 = \frac{1}{F_{ref}^2} \sum_{i=\lambda_1}^{\lambda_2} \lambda_i \sigma_{f(\lambda_i)}^2 R_X(\lambda_i)^2 (\lambda_i - \lambda_{i-1})^2 \tag{23}$$

(or the equivalent energy-counting version of this). Note that we do not account for error on F_{ref}, here. That's why we just use $\sigma_{f(\lambda_i)}$, which is the

error of your data spectrum.

If you're looking for *colors*, I've got your back there, too. For arbitrary color $X - Y$,

$$m_X - m_Y = -2.5 \log\left(\frac{F_X}{F_{ref,X}}\right) + 2.5 \log\left(\frac{F_Y}{F_{ref,Y}}\right). \tag{24}$$

We can propagate the error here, too. If you want, you can assume X and Y are independent, and do $\sigma_{X-Y} = \sqrt{\sigma_X^2 + \sigma_Y^2}$. However, we can be more rigorous and not assume independence. We're going to use $\sigma^2 = \mathbf{JCJ^T}$—the explanation of this formula is beyond the scope of this manual. $\mathbf{J}$ is the Jacobian of your spectrum (remember, your spectrum is a function!), and $\mathbf{C}$ is its covariance matrix. The following method will work *if you have an error for the flux in your data spectrum.*

For ease of calculations, we will treat $m_X - m_Y$ as a function of $f(\lambda_i)$ (our supernova spectrum, in discrete wavelength chunks). Then, for arbitrary color $X - Y$, the Jacobian is:

$$\mathbf{J}_{X-Y} = \begin{bmatrix} \frac{\partial m_{X-Y}}{\partial f(\lambda_0)} & \frac{\partial m_{X-Y}}{\partial f(\lambda_1)} & \cdots & \frac{\partial m_{X-Y}}{\partial f(\lambda_N)} \end{bmatrix} \tag{25}$$

where N is the last measured wavelength in the spectrum. The ith entry is:

$$\frac{\partial m_{X-Y}}{\partial f(\lambda_i)} = -\frac{1.09}{F_X} R_X(\lambda_i)(\lambda_i - \lambda_{i-1}) + \frac{1.09}{F_Y} R_Y(\lambda_i)(\lambda_i - \lambda_{i-1}) \tag{26}$$

Then, the covariance matrix is diagonal, and each entry is the spectrum error provided in the data:

$$\mathbf{C_{X-Y}} = \begin{bmatrix} \sigma^2_{f(\lambda_0)} & & & \\ & \sigma^2_{f(\lambda_1)} & & \\ & & \ddots & \\ & & & \sigma^2_{f(\lambda_N)} \end{bmatrix}. \tag{27}$$

Now, we use $\sigma^2 = \mathbf{JCJ^T}$, and boom, we have color error without assuming independence for the filters.

5.5 Things to watch out for

5.5.1 Response function units

Your response function will likely be normalized, but may be given in either normalized flux units or normalized photon counts. You need to know which units you have. If you need the other, don't fret—you can convert it to the other unit system. Let's say you have a response function in normalized photon units, but your spectrum is in energy flux units (ergs/cm^2/s/Å). It's best to convert the spectrum to photons. For each ith wavelength, you do:

$$f_\gamma(\lambda_i) = \frac{\lambda_i}{hc} f_E(\lambda_i), \tag{28}$$

where $h \approx 6.626 \times 10^{-27}$ ergs $\cdot$ s and $c \approx 3 \times 10^{18}$ is the speed of light in Å/s. Why convert the spectrum instead of the response function? Well, CCDs count photons, so it's ideal to do as much as possible in these units.

Don't forget to convert your errors, as well. Using Equation 41,

$$\sigma_{f_\gamma(\lambda_i)} = \frac{\lambda_i}{hc} \sigma_{f_E(\lambda_i)}. \tag{29}$$

5.5.2 Help! My results are unreasonable!

Did you check all of the following:

- Did you convert both your standard spectrum *and* your data spectrum from ergs to photons (or vice versa)?
- When you converted between ergs and photons, did you use the correct units for the constants? Remember, for ergs/cm^2/s/Å, use $h \approx 6.626 \times 10^{-27}$ ergs $\cdot$ s and $c \approx 3 \times 10^{18}$ Å/s.
- When you converted between ergs and photons, did you multiply by the conversion factor when you should have divided (or vice versa)?
- Does the wavelength range of your filter fall completely inside the wavelength range of your spectrum?
- Did you convert units when maybe you shouldn't have because you already did it?

6 Classifying spectra

You can use SNID or the in-browser option GELATO. Tutorials forthcoming.

7 Software and Models

7.1 SALT

7.1.1 What is SALT?

7.1.2 Models

SALT: a spectral adaptive light curve template for type Ia supernovae, J. Guy et al. 2005 [10]. This is the first SALT model. The model is trained on 34 SNe

with $z < 0.1$ in $UBVRI$ filters.

SALT2: using distant supernovae to improve the use of type Ia supernovae as distance indicators, J. Guy et al. 2007 [11]. SALT is retrained on data with $z \leq 1$.

Improved cosmological constraints from a joint analysis of the SDSS-II and SNLS supernova samples, M. Betoule et al. 2014 [12]. SALT2 is retrained on the data from [11], with the addition of SDSS-II data, for $z < 0.25$. SALT2.4 is born.

SALT3: An Improved Type Ia Supernova Model for Measuring Cosmic Distances, W. D. Kenworthy et al. 2021 [13]. SALT2.4 gets a facelift and becomes more open-sourcey.

SALT3-NIR: Taking the Open-source Type Ia Supernova Model to Longer Wavelengths for Next-generation Cosmological Measurements, J. D. R. Pierel et al. 2022 [14]. SALT3 gets upgraded to include NIR wavelengths.

7.1.3 How do I use SALT models?

SALT3 is in several places. You can use these models through:

- `sncosmo` (You can use `SNANA` models in `sncosmo`, as well.)
- `SNANA` (SALT2 only)

I haven't used `SNANA`, so I can't help you with that. However, Section 7.2 contains tips on using `sncosmo`.

7.2 sncosmo

`sncosmo` is pretty cool. It has a whole bunch of contributors. It's a Python package, useful for middle steps of supernova cosmology (e.g., fitting models). You can fit a lot of stuff with it, actually. `sncosmo` provides fit parameters for your data. You can use either spectroscopic or photometric data.

7.2.1 Fitting a light curve with photometry

You *must* input flux units into `sncosmo` in order for it to work. Trust me, I've tried using it with magnitudes because I was in denial that I would have to do a unit conversion (boo hoo). These should be *physical flux of your object, not flux of an object in a particular magnitude system* (see 4.2).

First, your data needs to be in an `astropy` table with columns `['date','flux','fluxerr','band','zp','zpsys']`.

7.2.2 Converting SALT parameter output to magnitude error along your entire light curve

`sncosmo` contains a function for converting your model fit to a light curve with covariance: `Model.bandfluxcov()`. You'd use it like this:

```
import sncosmo
import numpy as np
from astropy import Table
import pandas as pd

# Read in your data:
csv = pd.read_csv('SN2014J.csv')
# Define band and magnitude system:
band = 'B'
phase = np.arange(-10,30,0.1)
vega = sncosmo.get_magsystem('vega')

# Turn your pandas table into an astropy table
# so sncosmo can use it, and fit the model.
data = Table.from_pandas(csv)
model = sncosmo.Model(source='salt3')
result, fitted_model = sncosmo.fit_lc(data, model,
    ['t0', 'x0', 'x1', 'c'])

flux_lc, cov = fitted_model.bandfluxcov(band, phase)
```

where `flux_lc` is an array of the flux through your `band` at all phases in `phase`. Because I did *not* provide the zero point or zero point system as an argument (`'zp'` and `'zpsys'`, respectively), this is the physical flux of the object through the bandpass, `band` in photons/s/cm^2. `cov` is an $n \times n$ symmetric matrix, where n is the length of your `phase` array. Its entries contain the covariance between band fluxes at different phases.

However, *you need to provide* ***`zp`*** *and* ***`zpsys`*** *to* ***`Model.bandfluxcov()`*** *if you are converting to magnitudes, because magnitudes are a relative unit system.* You could also use the "hidden" function `Model._bandflux_rcov()` to calculate the covariances. This calculates the *relative* covariance—i.e., covariances that are *fractions* of the flux. So, if you do this, you need to remember to multiply your covariance matrix by the flux. You can take the diagonal entries of the covariance matrix output from `Model._bandflux_rcov()`, invoking the assumption that all phases are independent measurements, and convert these to magnitudes using Equation 41 (or, to skip doing derivatives yourself, Equation 9). Like so:

```
import sncosmo
import numpy as np
from astropy import Table
import pandas as pd

# Read in your data:
csv = pd.read_csv('SN2014J.csv')
# Define band and magnitude system:
band = 'B'
phase = np.arange(-10,30,0.1)
vega = sncosmo.get_magsystem('vega')
# See the section on zeropoint trouble
# in sncosmo for an explanation of this line:
zeropoint = csv['mag'] +
2.5*np.log10*(vega.band_mag_to_flux(csv['mag'], band)

# Turn your pandas table into an astropy table
# so sncosmo can use it, and fit the model.
data = Table.from_pandas(csv)
model = sncosmo.Model(source='salt3')
result, fitted_model = sncosmo.fit_lc(data, model,
    ['t0', 'x0', 'x1', 'c'])

flux_lc, cov = fitted_model.bandfluxcov(band, phase,
    zp=zeropoint, zpsys=vega)

flux_err = np.sqrt(cov.diagonal()) # Multiply flux_err by flux_lc
                                   # if you use _bandflux_rcov().
mag_err = np.sqrt(((2.5/np.log(10))**2)*(1/(flux_lc**2))
    *flux_err**2)
```

7.2.3 `helio_to_cmb()`

The below code is handy for converting your heliocentric redshift to CMB redshift. Honestly, I can't find its location in the sncosmo github right now, so I've copied and pasted it below:

```
import math
import numpy as np
from astropy.coordinates import SkyCoord

# From sncosmo:
def radec_to_xyz(ra, dec):
    # SUPERNOVA BOOTCAMP MANUAL AUTHOR ADDITION:
    # Modified to add the try/except statement
    try:
```

```
            x = math.cos(np.deg2rad(dec))
                * math.cos(np.deg2rad(ra))
            y = math.cos(np.deg2rad(dec))
                * math.sin(np.deg2rad(ra))
            z = math.sin(np.deg2rad(dec))
        except:
            coord = SkyCoord('%s %s' % (ra, dec),
                unit=(u.hourangle,u.deg))
            x = math.cos(np.deg2rad(coord.dec.degree))
                * math.cos(np.deg2rad(coord.ra.degree))
            y = math.cos(np.deg2rad(coord.dec.degree))
                * math.sin(np.deg2rad(coord.ra.degree))
            z = math.sin(np.deg2rad(coord.dec.degree))

        return np.array([x, y, z], dtype=np.float64)

    def cmb_dz(ra, dec):
        # See http://arxiv.org/pdf/astro-ph/9609034
        CMBcoordsRA = 167.98750000 # J2000
        CMBcoordsDEC = -7.22000000

        # J2000 coords from NED\n",
        CMB_DZ = 371000. / 299792458.
        CMB_RA = 168.01190437
        CMB_DEC = -6.98296811
        CMB_XYZ = radec_to_xyz(CMB_RA, CMB_DEC)
        coords_xyz = radec_to_xyz(ra, dec)
        dz = CMB_DZ * np.dot(CMB_XYZ, coords_xyz)

        return dz

    def helio_to_cmb(z, ra, dec):
        # Convert from heliocentric redshift to CMB-frame redshift.
        "    Parameters\n",
        "    ----------\n",
        "    z : float\n",
        "        Heliocentric redshift.\n",
        "    ra, dec: float\n",
        "        RA and Declination in degrees (J2000).\n",
        "    \"\"\"\n",

        dz = -cmb_dz(ra, dec)
        one_plus_z_pec = math.sqrt((1. + dz) / (1. - dz))
        one_plus_z_CMB = (1. + z) / one_plus_z_pec

        return one_plus_z_CMB - 1.
```

7.2.4 Help! I'm having zero point trouble.

First, make sure you're inputting *physical flux units*, not the flux of your object in a magnitude system. If you converted from magnitudes to flux, this is likely to be the issue (see section 4.2). So, what you need to do is take your reference spectrum back out of your magnitude.

Now, if you've done this, you can't forget to also convert your magnitude errors back to flux errors. It's the same equation as the error equation in 4.2. No funny business with the conversion back to physical units because that's just a constant offset, so the derivative from Equation 41 takes it out.

Here's some Python code to help you out with this fix:

```
import sncosmo
import numpy as np
from astropy import Table
import pandas as pd

# Read in your data:
csv = pd.read_csv('SN2014J.csv')
# Define band and magnitude system:
band = 'B'
vega = sncosmo.get_magsystem('vega')

# This is the line where you calculate the offset to
# remove the Vega spectrum:
zeropoint = data['mag'] +
    2.5*np.log10*(vega.band_mag_to_flux(data['mag'], band)

# Throw that offset into your table:
data['zp'] = pd.Series([zeropoint
    for x in range(len(data.index))])

# Calculate your flux error (absolute value because it's
# squared and then square-rooted):
data['fluxerr'] = abs((data['magerr']*np.log(10)/2.5)*
    (vega.band_mag_to_flux(data['mag'], band))

# Turn your pandas table into an astropy table
# so sncosmo can use it, and fit the model.
data = Table.from_pandas(csv)
model = sncosmo.Model(source='salt3')
result, fitted_model = sncosmo.fit_lc(data, model,
    ['t0', 'x0', 'x1', 'c'])
```

7.3 SNooPy

SNooPy [15, 16], written in Python by Chris Burns for the Carnegie Supernova Project, has a lot of handy functions. Broadly, it is a Python package that fits light curves, but it's broken up into a lot of separate functions, which can be very useful. You can run SALT2 and MLCS2k2 models via SNooPy, as well. Below, I'll list some of my favorite functions that are not-so-discussed in the documentation.

7.3.1 `get_dust_RADEC()` and `get_dust_sigma_RADEC()`

`get_dust_RADEC()` and `get_dust_sigma_RADEC()` query IRSA using a given RA and dec to get the Milky Way $E(B-V)$ and error in $E(B-V)$, respectively. These are located in `snpy.utils.IRSA_dust_getval`. They accept arguments for RA and dec, with the default dust map from [17]. They return two things: the result and a flag. The flag indicates that the function worked. You can throw this out.

Usage example:

```
from snpy.utils.IRSA_dust_getval import get_dust_RADEC,
    get_dust_sigma_RADEC

'''
SN 1987A coordinates from
http://simbad.u-strasbg.fr/simbad/sim-id?Ident=SN+1987A.
Note: At the time of writing this, the get_dust_RADEC()
function works, but get_dust_sigma_RADEC() is failing.
Otherwise, this code works as-is.
'''

ra, dec = 279.703427, -31.937066
mwreddening,_ = get_dust_RADEC(ra, dec, calibration='SF11')
e_mwreddening,_ = get_dust_sigma_RADEC(ra, dec,
    calibration='SF11')
mwreddening = mwreddening[0]
e_mwreddening = e_mwreddening[0]

print(f'The Milky Way reddening for SN 1987A is {mwreddening}
    +/- {e_mwreddening}.')
```

7.4 mpfit

`mpfit` is a very flexible Levenberg-Marquardt χ^2 minimization code. It can take error in both the x and y axes into account. It's useful to be familiar with! The Python code is located here. What I think you should do with this

is download this *.py file, and make it so it's importable. Then, write some functions that make it easier to use:

```
import os, sys
import numpy as np

# The next two lines assume that you are making this file
# in the same directory as the ``mpfit'' directory. i.e.,
# /home/this_file.py
# and
# /home/mpfit/mpfit/mpfit.py
dirname = os.path.dirname(os.path.abspath(__file__))
sys.path.append(os.path.join(dirname,'mpfit'))

from mpfit import mpfit

def Flin(x,p):
    """
    Set up linear function for mpfit to use.
    You can use any kind of function you want, though.
    The line is just an example.
    DO NOT call this.
    """
    y =  p[0] + p[1]*x
    return y

def myfunctlin(p, fjac=None, x=None, y=None, xerr=None,
    yerr=None):
    """
    Set up chisq to minimize for mpfit. This will accept any
    combination of xerr and yerr (i.e., you do not need either,
    or both).
    DO NOT call this.
    """
    # Parameter values are passed in "p".
    # If fjac==None then partial derivatives should not be
    # computed.  It will always be None if MPFIT is called
    # with the default flag.
    model = Flin(x, p)
    # Positive status value means MPFIT should continue,
    # negative means stop the calculation.
    status = 0

    # If you don't use the line function, you need
    # to change this and write your own chi^2 here.
    if xerr is not None and yerr is not None:
        return [status, np.sqrt((y-model)**2/(yerr**2 +
```

```
            (p[1]**2)*xerr**2))]
    elif xerr is None and yerr is not None:
        return [status, np.sqrt((y-model)**2/(yerr**2))]
    elif xerr is not None and yerr is not None:
        return [status, np.sqrt((y-model)**2/((p[1]**2)*xerr**2))]
    elif xerr is None and yerr is None:
        return [status, y-model]

def linfit(x,y,ex=None, ey=None, initial_guess=[0,1]):
    """
    Do the linear fit. USE THIS FUNCTION DIRECTLY IN YOUR CODE.
    i.e., you'd write something like:
    from this_file import linfit
    """
    # p0 are the initial conditions
    p0=np.array(initial_guess,dtype='float64')
    if ex is not None and ey is not None:
        fa = {'x':x, 'y':y, 'xerr': ex, 'yerr': ey}
    elif ex is None and ey is not None:
        fa = {'x':x, 'y':y, 'yerr': ey}
    elif ex is not None and ey is None:
        fa = {'x':x, 'y':y, 'xerr': ex}
    elif ex is None and ey is None:
        fa = {'x':x, 'y':y}
    m = mpfit(myfunctlin, p0, functkw=fa)
    return m.params, m.covar
```

8 Packaging your code in Python

This is possibly the most useful thing that I ever learned to do in graduate school. First of all, I'd like to give props to Code/Astro for being an incredible workshop where I learned this. This tutorial will be the barest, most barebones way to package your stuff. This will not cover things like unit tests, licenses, or documents.

Let's say we have a file, `calculate_magnitudes.py`. It looks like this:

```
import numpy as np

def calc_mag(f):
    return -2.5*np.log10(f)
```

and we want to be able to import it into a different file because it's an enormous collection of functions and you don't want to copy and paste that monster (just

use your imagination, okay?). Let's name our package `magpy`. You will make a folder with the following structure and contents:

```
> magpy
    > calculate_mag
        calculate_magnitudes.py
        __init__.py
    README.md
    requirements.txt
    setup.py
```

Let's talk about what each of these are, and what goes in each of them.

`> magpy` — This folder contains everything. It is the highest-level directory in the package.

`> calculate_mag` — This folder contains your code. It may also have accompanying data files.

`calculate_magnitudes.py` — This is your code. Your functions and tools you want to use exist in here.

`__init__.py` — This file is run when you first import the package. If you don't want to write a long string of stuff like `from magpy.calculate_mag.calculate_magnitudes import calc_mag`, write the following in this file:

```
from .calculate_magnitudes import *
```

This line imports everything in `calculate_magnitudes.py`. So, you can just write `import calculate_mag` and then in your code call `calc_mag()` directly.

`README.md` — This describes the software and provides instructions on how to use it. Maybe it has citations, maybe it has a quick code example, maybe it has your name.

`requirements.txt` — This lists dependencies for your package. For this example, the only thing in this file is:

```
numpy
```

`setup.py` — You need this to make `python setup.py install` work! The most basic setup file looks like:

```
from setuptools import setup, find_packages

setup(name='magpy',
    version='v1.0.0',
    packages=find_packages())
```

Now, you can install your glorious tools using `pip install -e .` or `python setup.py install` from your terminal in the `magpy` directory. Throw it on github so you can brag about it and put it on your CV, too.

9 Statistics Stuff

I guess you could argue this is outside the scope of a supernova manual, but these are things I use all the time. Plus, it's my darn manual, so I get to put what I want in here.

9.1 Weighted expectation value, variance, and covariance

Expectation value, unweighted by data error:

$$E[X] = \sum_i x_i p_i, \tag{30}$$

where x_i is your data and p_i is the probability of that value. Now, if you want to use your data's error instead of the probability (which is hard/weird to quantify):

$$E_w[X] = \frac{\sum_i x_i/(\sigma_{x_i}^2)}{\sum_i (1/\sigma_{x_i}^2)}, \tag{31}$$

where σ_{x_i} is the error for data point x_i.
Great, then what's the **variance of a weighted expectation value** (mean)? Unweighted, variance is defined as:

$$\mathrm{Var}[X] = E[(X-\mu)^2] = \sum_i p_i (x_i - \mu)^2, \tag{32}$$

where μ is the mean of the data and p is the probability of that x_i occurring. So, we calculate the weighed variance with this formula, but using the weighted mean (a.k.a. expectation value) from Equation 31, and change p_i to $w_i = 1/\sigma_i^2$:

$$\mathrm{Var}_w[X] = E_w[(X-\mu_w)^2] = \frac{1}{\sum_i \sigma_{x_i}^2} \sum_i \frac{1}{\sigma_{x_i}^2} (x_i - E_w[X])^2) \tag{33}$$

Now, on to **weighted covariance**. Covariance is defined as

$$\mathrm{cov}(X,Y) = E[(X - E[X])(Y - E[Y])] \tag{34}$$

So, our weighted covariance is going to be this formula, but using weighted variance and expectation values:

$$\mathrm{cov}_w(X,Y) = E[(X - E_w[X])(Y - E_w[Y])] \tag{35}$$

The tricky part here is how we handle the errors. Using the error propagation formula (Equation 41) on $(X-E[X])(Y-E[Y])$ and then taking the reciprocal of this, we can weight by

$$w_i = \frac{1}{(y_i - E_w[Y])^2\sigma_{x_i}^2 + (x_i - E_w[X])^2\sigma_{y_i}^2} \tag{36}$$

Thus, weighted covariance is

$$\mathrm{cov}_w = \frac{1}{\sum_i w_i}\sum_i w_i(x_i - E_w[X])(y_i - E_w[Y]) \tag{37}$$

9.2 Weighted Pearson correlation coefficient

The Pearson correlation coefficient measures how linearly correlated two datasets are. It ranges from $[-1,1]$, where -1 is perfectly negatively linearly correlated, 1 is perfectly positively linearly correlated, and 0 is no correlation. Your correlation coefficient, weighted or unweighted, is:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y}, \tag{38}$$

where $\mathrm{cov}(X,Y)$ is the covariance between datasets X and Y, and σ is the error for a dataset. In order to make this weighted, you calculate the weighted covariance for the numerator, and use weighted variances for the denominator. In terms of things we've discussed earlier in this chapter:

$$\rho_{X,Y,w} = \frac{\mathrm{cov}_w(X,Y)}{\sqrt{\mathrm{Var}_w(X)\mathrm{Var}_w(Y)}} \tag{39}$$

That's great and all, but what about the significance of this? What's the p-value? We can calculate this with a two-sided t-test, where the test statistic is

$$t = \frac{\rho_{X,Y}\sqrt{N-2}}{\sqrt{1-\rho_{X,Y}^2}}. \tag{40}$$

$\rho_{X,Y}$ is your Pearson correlation coefficient (weighted or unweighted), N is the number of data points, and 2 represents the free parameters in the fit (i.e., $N-2$ is the degrees of freedom in the fit). To get the p-value, you have three options: 1. Do a painful integral 2. Use a table 3. Use `scipy.special.stdtr` in Python, which does the painful integral for you. I will not be discussing options 1 and 2, because they stink. Option 3 is best. Use `stdtr()` like this: `2*stdtr(dof,-|t|)`, where `dof` is your degrees of freedom, $(N-2)$, and `t` is the test statistic. You slap the absolute value signs on and multiply the final p-value by 2 because you're integrating between two values in the t distribution.

9.3 Error Propagation

In order to propagate error, you need to know the functions that describe your model, because you're going to have to take derivatives. The simplest way to propagate error for a function with n variables is: $f(x_0, x_1, ..., x_n)$ is:

$$\sigma_f = \sqrt{\sum_{i=0}^{n} \left(\frac{\partial f}{\partial x_i}\sigma_{x_i}\right)^2}, \tag{41}$$

where σ_{x_i} is the error for variable x_i. Note that this formula *assumes the variables are all independent.*

9.4 Least Squares and Minimizing χ^2

In general, when minimizing χ^2 to fit a model,

$$\chi^2 = \sum_{i=0}^{k} \left(\frac{X_i - \mu_i}{\sigma_i}\right)^2 \tag{42}$$

where X_i is the data, μ_i is the model, and σ_i is the error for the data. k is the number of data points.

Now, use Python to minimize this function. You can use any minimization algorithm you want. Personally, I like least squares. You can use `pycmpfit` or `mpfit` (see Section 7.4). Both will work, but `pycmpfit` is installable via pip. If you want to use `mpfit`, you can do something like I did here, which is copied in to Section 7.4 (and you can `git clone` and then `pip install -e .` the entire `LaurensTools` package, if you want).

9.4.1 Example: Fitting a line

We have a model that we want to fit to our data,

$$y(x) = mx + b, \tag{43}$$

where m is the slope and b is the $y-$intercept. These are our parameters to be fit to our data. We want to find out what values of m and b fit the data best. So, our χ^2 *without* considering data error is:

$$\chi^2 = \sum_{i=0}^{k} (X_i - y(x_i))^2, \tag{44}$$

where X_i is the ith data point, corresponding to independent variable x_i, and $y(x_i)$ is the model at point x_i. Expanded, we have:

$$\chi^2 = \sum_{i=0}^{k} (X_i - (mx_i + b))^2, \tag{45}$$

If we want to consider error in both x and y:

$$\chi^2 = \sum_{i=0}^{k} \frac{(X_i - (mx_i + b))^2}{\sigma_y^2 + m^2\sigma_x^2} \tag{46}$$

9.4.2 Example: Fitting a Hubble diagram

Let's say we want to use the following model for predicting distance using SNe Ia [18]:

$$\mu = m_B - M - \alpha(C - C_{avg}) - \delta(\Delta m_{15} - \Delta m_{15,avg}) \tag{47}$$

The, the χ^2 to minimize is

$$\chi^2 = \sum_i \frac{(\mu_i - (m_{Bmax,i} - M - \alpha(C_i - C_{avg}) - \delta(\Delta m_{15,i} - \Delta m_{15,avg})))^2}{\sigma_{vpec,i}^2 + \sigma_{Bmax,i}^2 + (\alpha\sigma_{C,i})^2 + (\delta\sigma_{\Delta m_{15},i})^2} \tag{48}$$

i represents each SN in the sample, and σ_{vpec} is error in peculiar velocity, with $v_{pec} = 300$ km s^{-1}.

10 Observing

Whelp, if you're reading this, I can only assume it's because you're about to irreparably mess up your week by staying up until heinous hours of the morning. Enjoy.

10.1 Making your observing plan

10.1.1 Exposure time

I guarantee your telescope/instrument has an exposure time/integration time calculator. Go find it.

10.1.2 Signal-to-noise ratio

The signal-to-noise ratio (S/N) compares your signal to your noise. $(S/N) < 1$ means your noise is the predominant source, and $(S/N) > 1$ means your signal (or, the thing you actually *want* to see in your observations) is the predominant source. It's defined like:

$$S/N = \frac{S_{obj}}{S_{noise}} \tag{49}$$

The higher your (S/N), the better the observations. Note that your magnitude error σ_m scales with $1/(S/N)$, i.e., higher (S/N) means lower magnitude errors. Some key values to remember are: $(S/N) = 10 \implies \sigma_m \sim 0.1 \times m$ and

$(S/N) = 100 \implies \sigma_m \sim 0.01 \times m$. So... what does this mean? It means that $(S/N) \sim 10$ is probably the lowest you'll want to go for a given observation. If you need a quick spectrum, it's fine, you can do coarse measurements of large absorption lines here. You will most often encounter (S/N) when you're deciding how long your exposure times need to be.

Unless you're building an exposure time calculator, you don't really need all the details on this equation, so I defer those detailed explanations to The Greater Internet$^{(TM)}$ because the point of this manual is to help you actually *use* nebulous (lol) concepts.

10.2 Observing logs

If you're observing, you should keep a log that tracks what you did, when you did it, weather conditions, seeing, etc. If something goes wrong, write it down. This may be important down the line! Nothing is too unimportant to jot down. If you're in doubt, just write it down.

10.3 There's extra time! Or I ran out of time!

10.3.1 Extra time

If your observing program provided extra targets, take a look at their RAs before you begin the night. Look at the schedule to see when you're observing objects with similar RA. If you're ahead of schedule while you're doing these, go ahead and add in the extra! If you don't know what objects to add, you can always repeat observations. No one will ever be mad about this. The most important thing is to check that the RA for the additional observation makes sense for the current time, and the easiest way to do this is to make sure it's close to the RA of the object(s) you most recently observed.

10.3.2 Ran out of time

Nothing you can really do about this. If you've hit morning and you haven't observed everything, that's that. If it's the middle of the night and you're behind schedule, you can make some decisions to skip targets. If you know the weather is going to be bad, it doesn't hurt to find out which objects are lowest priority. Always make a note in the observing log if you chose to skip something, and write down why. It's probably not your fault—the schedule may have been a bit off, or weather forced you to close the dome. No one will criticize you for this! These things are out of your control.

10.4 Remote

10.4.1 Before you begin the night...

There are a few things you need to do before you begin the night. First, make sure you have all the necessary software installed and you know how to access all in-browser GUIs. You might need:

- A VPN client. I use Cisco AnyConnect (at the time of writing this, Texas A&M provided this software, so any instructional VPN-related screenshots will be using Cisco AnyConnect on an Ubuntu system). If you're an A&M affiliate, you can get it through connect.tamu.edu. You *must* have Duo set up to do this. I spent like, two weeks trying to figure out why I couldn't get the software until I realized it was because I wasn't getting push notifications from Duo on my phone.
- A VNC viewer. Real VNC is nice and it is free.
- A stable internet connection. If you're doubtful of your home internet, cut your losses and go to your office. Logging back in to everything every time you lose connection will be extremely annoying.
- Something(s) to keep yourself awake. Food, caffeine, music, games, etc. Observing is your priority though, so whatever it is, make sure you can drop it if you need to. For example, I do not recommend League of Legends, but I do recommend Stardew Valley. A lot of people get work done during the night, but personally, I am incapable of being useful past 10:00 PM.

Second, it helps to organize your scripts/plan in advance. Do whatever you want, but I like to put scripts in separate folders for $\sim$ 1 hr blocks of time so I don't have to think about where/when all my targets are at night when my brain is running on steam.

10.4.2 CTIO/DECam

You should always know what the local time is. For CTIO, the time is the same as the time in Santiago, Chile.

In general, in this section, usernames/passwords/addresses will be obscured or not provided. You should get these from support. If you need your proposal ID, check the schedule for the 4m telescope at CTIO.
First, make sure you have installed:

- A VPN client
- A VNC viewer

You can join the Zoom call any time between now and when you need to request observer permissions. If you have trouble with something, join the Zoom call and ask for help!

The first thing you need to open in your computer is your VPN. You need to do this to access the network at CTIO. Log in like in Figure 5, but get your address, group, username, and password from support.

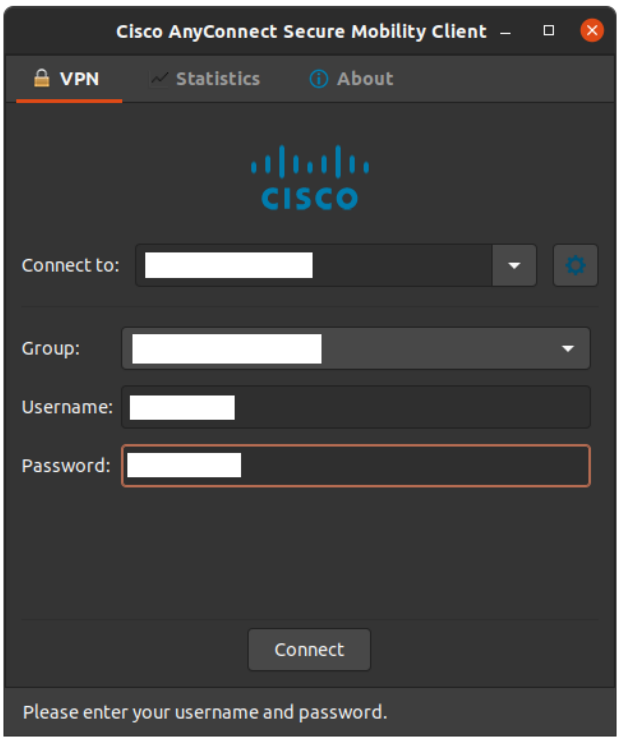

Figure 5: This is what your VPN window should look like for DECam, if you're using Cisco AnyConnect.

Open up your VNC viewer. Enter the address and password. RealVNC will save this information for next time. The VNC viewer allows you to remotely use a computer at CTIO.

I think it's also nice to open the CTIO external webcams. It's the only view of the sky at CTIO you'll get all night. Plus, it's a quick way to check the weather. A slightly less quick, but more quantitative way to check the weather is to use the CTIO Site Environmental Conditions page.

Now, open the SISPI GUIs. You'll see a bunch of options, like in Figure 6. If you're not on the VPN, you won't be able to access this page. Open one of apps in a new tab and enter the username and password provided by support.

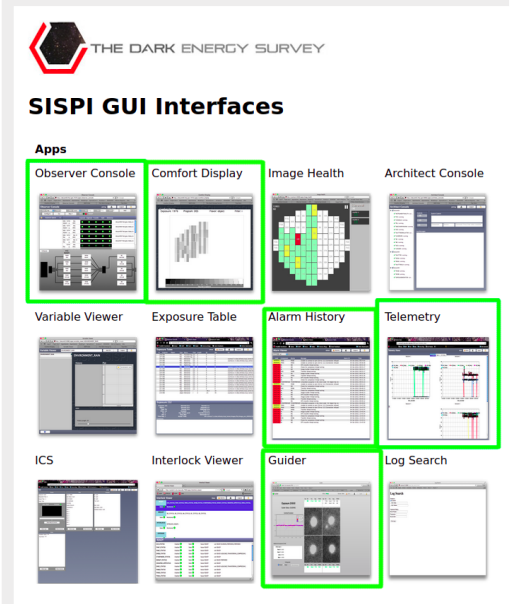

Figure 6: The SISPI GUIs, with the important ones (or so I think) highlighted in green.

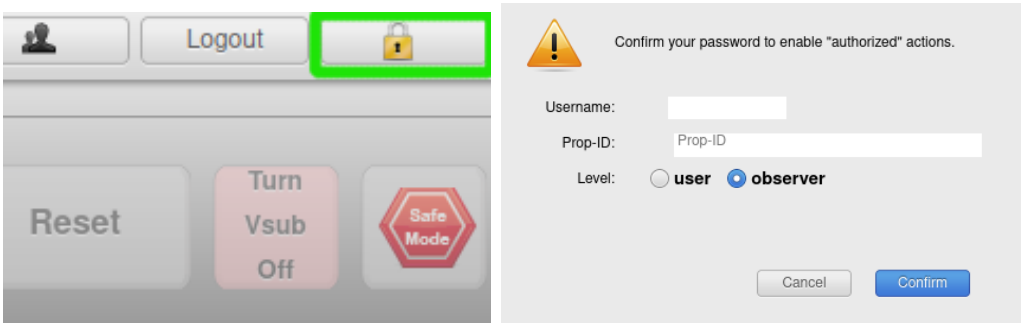

Figure 7: *Left*: The location to click to begin requesting observer permissions. *Right*: The pop-up window to request observer permissions.

If it's not working and you've copied and pasted the login info in to the text boxes, try typing it in by hand. This has been an issue for me in the past, I'm unsure if it's inherent to the software, or if it's a browser issue, or something else.

In the Observer Console app, which you access from the SISPI GUI Interfaces page, the first thing you need to do is request observer permissions. In the upper right-hand corner, there's a padlock button (see the left panel of Figure 7). Click this. Then, a window will pop up like in the right panel of Figure 7. Enter your proposal ID, change the "level" button from "user" to "observer", and click "confirm". The username will already be entered. Kindly ask your support staff on Zoom to grant you observer permissions.

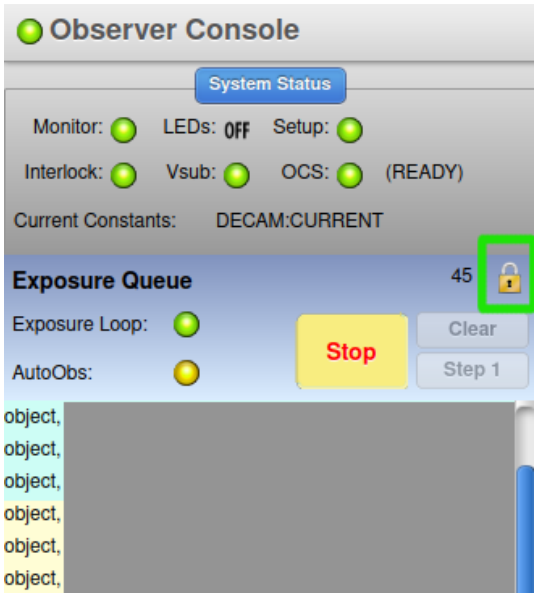

Figure 8: The exposure queue, with the padlock icon to allow deleting/shuffling of exposures as-needed. Target information from this screenshot is greyed out. To the left of the padlock, "45" indicates that there are 45 exposures currently loaded into the queue. Underneath these icons, the "Stop" button (which is often red, not yellow, I don't know why it's yellow here) will interrupt the exposure loop. When pressed, it will change to "Go". This flip-flops depending on if the exposure loop is running.

Once you're in, there are a few things you can do. First, update the Observers, Proposal ID, Program, and Investigator information if it hasn't already been updated. You'll find these in the "System Control" tab along the top of the screen. Click "Edit", edit the text boxes, and then click "Save". In the "Exposure Control" tab, you upload the actual json scripts. Click "Load exposure script", "Browse..." and then "Submit" (this may appear as "Sub..."). It'll appear in the exposure queue as a block of yellow, green, or blue exposures. The colors don't mean anything, they're just to differentiate between which exposures belong to which scripts.

Once your stuff is in the exposure queue, you can move around or delete the exposures as needed. Click the padlock icon, highlighted in Figure 8, to unfreeze the queue. Now, you can delete things or move them around if you need to. This is uncommon, but it happens. When you're done, *always re-lock it.* To the left of the padlock, there are numbers that switch between integers and a timer. The timer is the amount of exposure time remaining in the queue. The integer is the number of exposures remaining in the queue.

In the Telemetry Viewer app, I like to look at the Pointing and Image Health tabs. All plots indicate the values calculated for the *previous* image. They are

not updated in real-time. The Pointing tab has pointing and center offsets plots—you'll want to keep an eye on these to make sure these don't drift too far from center. Image Health has a seeing plot, which shows the... seeing. Write this down in your observing log every so often. You don't have to be super precise. This number gets recorded in the FITS header anyway.

The Comfort Display SISPI GUI is also important. You should look at all exposures here, in addition to some in the VNC viewer to check for image saturation. If your background counts are $> 40,000$, the detector is definitely saturated. **You must stop saturated observations immediately**. They can damage the instrument. $> 20,000$ is maybe not instrument-damaging, but they're a waste of time. Expect the $g-$band to be saturated during bright time, and the $r-$band if you're pointing close to the Moon or have a very long exposure (> 100s) in bright time. You CAN stop an exposure in the middle and look at counts before continuing if you're concerned it may saturate the detector.

You'll want to keep track of the Alarm History GUI as well; if there's a warning, you should figure out what's causing it. Finally, open the Guider app. This will tell you if there are guiding issues, which can arise if you're in a crowded field.

Let's talk about the center offset correction. In the VNC viewer, if a terminal is not already open, open a terminal. Type `observer`, hit enter, then `center`, then hit enter. This will print the center offset for the previous image. If it's large, kindly ask your support staff in the Zoom call for a pointing correction. Listen to their instructions about hitting "Stop" and "Go" in the exposure queue (the correction can't be made in the middle of an exposure—you need to interrupt the exposure loop). I usually ask for a correction around a 10" offset. If you're near the end of observing an object, don't worry about it, wait until the telescope slews to see if it corrects itself. If not, go ahead and ask.

Now that I've said all that, I'll go over using the terminal in the VNC viewer. You'll use a package called Kentools, which you enter in the terminal by typing "observer". Exit by typing "exit". Like this:

```
# Open Kentools:
observer2> observer
# Close Kentools:
prompt> exit
```

That's it! It'll pull up a long list of commands, which I will not copy and paste here. To check centering and seeing, do this:

```
# Open Kentools:
observer2> observer
# List commands:
```

```
prompt> commands
# Check centering for previous image:
prompt> center
# Check seeing for the previous image:
prompt> seeingall
```

You can open the images you're taking and look at them in DS9! This is also how you check counts.

```
# From Kentools, check image inventory
prompt> inv
# Load the S4 CCD for the previous image (default).
prompt> load
```

DS9 will pop up with the image. You can check the background counts by hovering your mouse over any location in the image and reading what's in the "value" box underneath the "File" and "Object" information fields. Here's another example:

```
# From Kentools, check image inventory
prompt> inv
# I picked exposure number 1156247, and load the N5 CCD.
prompt> load 1156247 36
```

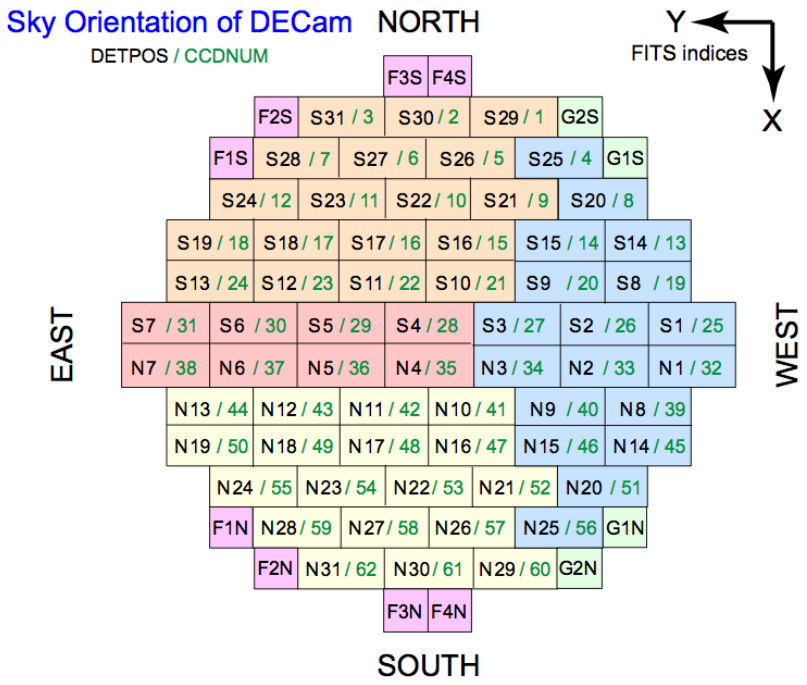

Figure 9: Alphanumeric codes for each DECam CCD.

You can get the exposure number from several places. You can type "inv" in Kentools to get an inventory of the exposures from that night. The first column is the exposure number. You can also get the most recent exposure number from the dialogue box in the exposure console, or the upper left-hand corner of the comfort display GUI. You can get the CCD code from Figure 9. The N5 CCD has a green 36 written next to it, so I type "36" to specify that I want to look at that particular CCD. You can also look at the ENTIRE frame with "bigload". Same deal as "load", but you don't need to specify the CCD because it loads all the CCDs: `bigload [expnum]`.

At the end of the night, you need to fill out the end-of-night report. If you observed during the first half of the night, you need to create it. If you observed during the second half, you will get the link from the first half observer (unless they didn't start it. Then you need to make it). Get the login credentials from support. It is very important to ask the telescope operator who needs to be reported in the night log! Don't be afraid to ask.

At the end of the night, grab the inventory file. In the terminal in your VNC viewer (or if you SSH'd in via a terminal on your computer):

```
# From Kentools:
prompt> godb
prompt> invPrint
prompt> qcInvPrint
```

Then, on your local computer, i.e., in a terminal *without* the SSH connection,

```
scp DECamObserver@observer2.ctio.noao.edu:/user/DECamObserver/
    yyymmdd.qcinv .
```

where `yyymmdd` is year, month, day. Also, don't drop the period at the end of this command! You'll need to put in the SSH password, and then a file called `yyyymmdd.qcinv` will appear in the folder from where you ran the `scp` command.

11 Theory

Haha, we weren't going to get away without a theory section. Haha. hahaha-hahaha.

11.1 Deflagration vs. Detonation

11.2 Arnett's rule

11.3 Important publications

Early Supernova Luminosity, Colgate and McKee 1969 [19]. First paper that shows that ^{56}Ni is a "predominant end product of thermonuclear processes", and thus may be synthesized in large quantities in the ejecta of SNe Ia (but they were just called "Type I" back then). Astronomy is introduced to the decay sequence of ^{56}Ni: $^{56}\text{Ni} \rightarrow {}^{56}\text{Co} \rightarrow {}^{56}\text{Fe}$. ^{56}Co decay is mentioned, but no conclusions about it are drawn.

On the Theory of Type I Supernovae, Arnett 1979 [20]. Uses the idea from Colgate and McKee 1969 [19] to show that, actually, yeah, ^{56}Ni decay is a totally reasonable explanation for Type I supernova light curves. This decay entirely explains the bolometric light curve near peak brightness, and at later times ($t \gtrsim 100$ d), the exponential nature of the light curve can be explained by ^{56}Co decay.

Type I supernovae. I - Analytic solutions for the early part of the light curve, Arnett 1982 [21].

Seeing the Collision of a Supernova with its Companion Star, Kasen 2010 [22].

Acknowledgements

The author thanks the following people for useful comments: Peter Brown, D'Arcy Kenworthy, Antonella Palmese, Nick Suntzeff, and Jiawen Yang.

References

[1] Edward L. Fitzpatrick. Correcting for the effects of interstellar extinction. *Publications of the Astronomical Society of the Pacific*, 111(755):63–75, Jan 1999. arXiv: astro-ph/9809387.

[2] Erik R. Peterson, W. D'Arcy Kenworthy, Daniel Scolnic, Adam G. Riess, Dillon Brout, Anthony Carr, Hélène Courtois, Tamara Davis, Arianna Dwomoh, David O. Jones, Brodie Popovic, Benjamin M. Rose, and Khaled Said. The Pantheon+ Analysis: Evaluating Peculiar Velocity Corrections in Cosmological Analyses with Nearby Type Ia Supernovae. , 938(2):112, October 2022.

[3] Ryan Foley, Daniel Scolnic, Armin Rest, Sourav Jha, yen-chen Pan, A. Riess, P. Challis, Kenneth Chambers, D. Coulter, K. Dettman, M. Foley,

O. Fox, M. Huber, David Jones, C. Kilpatrick, Robert Kirshner, A. Schultz, M. Siebert, H. Flewelling, and Mervi Willman. The foundation supernova survey: Motivation, design, implementation, and first data release. *Monthly Notices of the Royal Astronomical Society*, 475, 11 2017.

[4] B. W. Rust. *The Use of Supernovae Light Curves for Testing the Expansion Hypothesis and Other Cosmological Relations.* PhD thesis, Oak Ridge National Laboratory, Tennessee, January 1974.

[5] Iu. P. Pskovskii. Light curves, color curves, and expansion velocity of type I supernovae as functions of the rate of brightness decline. , 21:675, December 1977.

[6] M. M. Phillips. The Absolute Magnitudes of Type IA Supernovae. , 413:L105, August 1993.

[7] M. M. Phillips, Paulina Lira, Nicholas B. Suntzeff, R. A. Schommer, Mario Hamuy, and José Maza. The Reddening-Free Decline Rate Versus Luminosity Relationship for Type IA Supernovae. , 118(4):1766–1776, October 1999.

[8] Adam G. Riess, Alexei V. Filippenko, Peter Challis, Alejandro Clocchiatti, Alan Diercks, Peter M. Garnavich, Ron L. Gilliland, Craig J. Hogan, Saurabh Jha, Robert P. Kirshner, B. Leibundgut, M. M. Phillips, David Reiss, Brian P. Schmidt, Robert A. Schommer, R. Chris Smith, J. Spyromilio, Christopher Stubbs, Nicholas B. Suntzeff, and John Tonry. Observational evidence from supernovae for an accelerating universe and a cosmological constant. *The Astronomical Journal*, 116(3):1009–1038, sep 1998.

[9] L. Galbany, M. E. Moreno-Raya, P. Ruiz-Lapuente, J. I. González Hernández, J. Méndez, P. Vallely, E. Baron, I. Domínguez, M. Hamuy, A. R. López-Sánchez, M. Mollá, S. Catalán, E. A. Cooke, C. Fariña, R. Génova-Santos, R. Karjalainen, H. Lietzen, J. McCormac, F. C. Riddick, J. A. Rubiño-Martín, I. Skillen, V. Tudor, and O. Vaduvescu. SN 2014J at M82: I. A middle-class type Ia supernova by all spectroscopic metrics. *Monthly Notices of the Royal Astronomical Society*, 457(1):525–537, oct 2015.

[10] J. Guy, P. Astier, S. Nobili, N. Regnault, and R. Pain. SALT: a spectral adaptive light curve template for type Ia supernovae. , 443(3):781–791, December 2005.

[11] J. Guy, P. Astier, S. Baumont, D. Hardin, R. Pain, N. Regnault, S. Basa, R. G. Carlberg, A. Conley, S. Fabbro, D. Fouchez, I. M. Hook, D. A. Howell, K. Perrett, C. J. Pritchet, J. Rich, M. Sullivan, P. Antilogus, E. Aubourg, G. Bazin, J. Bronder, M. Filiol, N. Palanque-Delabrouille, P. Ripoche, and V. Ruhlmann-Kleider. SALT2: using distant supernovae to improve

the use of type Ia supernovae as distance indicators. , 466(1):11–21, April 2007.

[12] M. Betoule, R. Kessler, J. Guy, J. Mosher, D. Hardin, R. Biswas, P. Astier, P. El-Hage, M. Konig, S. Kuhlmann, J. Marriner, R. Pain, N. Regnault, C. Balland, B. A. Bassett, P. J. Brown, H. Campbell, R. G. Carlberg, F. Cellier-Holzem, D. Cinabro, A. Conley, C. B. D'Andrea, D. L. DePoy, M. Doi, R. S. Ellis, S. Fabbro, A. V. Filippenko, R. J. Foley, J. A. Frieman, D. Fouchez, L. Galbany, A. Goobar, R. R. Gupta, G. J. Hill, R. Hlozek, C. J. Hogan, I. M. Hook, D. A. Howell, S. W. Jha, L. Le Guillou, G. Leloudas, C. Lidman, J. L. Marshall, A. Möller, A. M. Mourão, J. Neveu, R. Nichol, M. D. Olmstead, N. Palanque-Delabrouille, S. Perlmutter, J. L. Prieto, C. J. Pritchet, M. Richmond, A. G. Riess, V. Ruhlmann-Kleider, M. Sako, K. Schahmaneche, D. P. Schneider, M. Smith, J. Sollerman, M. Sullivan, N. A. Walton, and C. J. Wheeler. Improved cosmological constraints from a joint analysis of the SDSS-II and SNLS supernova samples. , 568:A22, August 2014.

[13] W. D. Kenworthy, D. O. Jones, M. Dai, R. Kessler, D. Scolnic, D. Brout, M. R. Siebert, J. D. R. Pierel, K. G. Dettman, G. Dimitriadis, R. J. Foley, S. W. Jha, Y. C. Pan, A. Riess, S. Rodney, and C. Rojas-Bravo. SALT3: An Improved Type Ia Supernova Model for Measuring Cosmic Distances. , 923(2):265, December 2021.

[14] J. D. R. Pierel, D. O. Jones, W. D. Kenworthy, M. Dai, R. Kessler, C. Ashall, A. Do, E. R. Peterson, B. J. Shappee, M. R. Siebert, T. Barna, T. G. Brink, J. Burke, A. Calamida, Y. Camacho-Neves, T. de Jaeger, A. V. Filippenko, R. J. Foley, L. Galbany, O. D. Fox, S. Gomez, D. Hiramatsu, R. Hounsell, D. A. Howell, S. W. Jha, L. A. Kwok, I. Pérez-Fournon, F. Poidevin, A. Rest, D. Rubin, D. M. Scolnic, R. Shirley, L. G. Strolger, S. Tinyanont, and Q. Wang. SALT3-NIR: Taking the Open-source Type Ia Supernova Model to Longer Wavelengths for Next-generation Cosmological Measurements. , 939(1):11, November 2022.

[15] Christopher R. Burns, Maximilian Stritzinger, M. M. Phillips, ShiAnne Kattner, S. E. Persson, Barry F. Madore, Wendy L. Freedman, Luis Boldt, Abdo Campillay, Carlos Contreras, Gaston Folatelli, Sergio Gonzalez, Wojtek Krzeminski, Nidia Morrell, Francisco Salgado, and Nicholas B. Suntzeff. The Carnegie Supernova Project: Light-curve Fitting with SNooPy. , 141(1):19, January 2011.

[16] Christopher R. Burns, Maximilian Stritzinger, M. M. Phillips, E. Y. Hsiao, Carlos Contreras, S. E. Persson, Gaston Folatelli, Luis Boldt, Abdo Campillay, Sergio Castellón, Wendy L. Freedman, Barry F. Madore, Nidia Morrell, Francisco Salgado, and Nicholas B. Suntzeff. The Carnegie Supernova Project: Intrinsic Colors of Type Ia Supernovae. , 789(1):32, July 2014.

[17] Edward F. Schlafly and Douglas P. Finkbeiner. Measuring Reddening with Sloan Digital Sky Survey Stellar Spectra and Recalibrating SFD. , 737(2):103, August 2011.

[18] Robert Tripp. A two-parameter luminosity correction for Type IA supernovae. , 331:815–820, March 1998.

[19] Stirling A. Colgate and Chester McKee. Early Supernova Luminosity. , 157:623, August 1969.

[20] W. D. Arnett. On the theory of type I supernovae. , 230:L37–L40, May 1979.

[21] W. D. Arnett. Type I supernovae. I - Analytic solutions for the early part of the light curve. , 253:785–797, February 1982.

[22] Daniel Kasen. Seeing the Collision of a Supernova with Its Companion Star. , 708(2):1025–1031, January 2010.