Last revision: February 9, 2004

Lightcurve Software Documentation/Proposal

Natalia Kuznetsova, Rob Knop

1. Introduction

This note describes the current light curve fitting software written by Rob. Note that I assume that the images are already reduced and cosmic-ray rejected (the reduction part of the analysis will require separate documentation!).

2. Technical Details: Sequence of Steps

Currently, the steps involved in extracting the light curve are:

- The first step is creating a parameter file that will be used for creating the light curve. The parameters that go into this file are described on Rob's Web page (http://brahms.phy.vanderbilt.edu/deepsearch/scp/deepsoft/hstltcv/index.html). The easiest way to obtain these parameters is to run `testfitswidget` in deeplib/src. The main advantage of using this image viewer is that it has the same (0,0) as the rest of the software (lower left corner).

- Run `hstltcv` (the package is actually called `simplehstltcv` and it is available from Rob's Web page). It is descibed in more detail in Section 3. This package requires an input "param" file described above. Note that some of the input parameters are set up in the code itself (`simplehstltcv.cc`).

  Apart from creating a light curve, the package also does two different versions of aperture photometry. The output is three ASCII files with extensions psfresults, aperbgresults, and aperringresults. The actual light curve is contained in the "psfresults" file. The other two files contain the output of the photometry algorithms (*to be described!*.

- The .psfresults file made in the previous step is not quite in the format expected by the fitting programs, so there are a few simple ASCII file manipulations involved:

  ```
  ./makehstV3_2.perl filter sn_filter.psfresults > sn.filter.version#_noinv
  ./invertcorrma sn.filter.version#_noinv > sn.filter.version#
  ```

- The next steps involve color-correcting and fitting the lightcurve. Color-correcting is actually a two-step process, since we need to apply both an "S-correction", which is basically reducing an HST filter to a standard ground-based filter, and a "K-correction", which relates supernova parameters in its rest frame and in the observed frame. The problem with these two steps is that they are inter-related: one can't color-correct the supernova without knowing some of its fitted parameters, such as stretch, and one can't correctly fit a supernova that has not been color-corrected. The solution is to perform both steps iteratively using `iterator.perl`, starting with a stretch of 1.0 and $E(B-V)_{\rm host} = 0$. Other parameters we should know are t_{max} (??), *col* (??), z, and $E(B-V)_{\rm MilkyWay}$ (the latter we can look up in http://nedwww.ipac.caltech.edu/ if you know the name – *which name do they mean though????*). The iterator script does the following:

- First, it sets up all the necessary supernova parameters (either from the command line, or from the hard-coded parameters, or from parsing an input "supernova.dat" file), as well as a number of directories that will be used in the calculations.
- Then it invokes `makesnminuitK`, which lives in the colv3 directory. This is a program that makes a K-correction file for SNMinuit. It takes the following input parameters: rest filter (e.g., u), observed filter (e.g., r), redshift, stretch, Host E(B-V), Milky Way E(B-V), RV to use for the MW reddening law, and RV to use for the host reddening law. *It is not quite clear to me what the last parameter in the input file for makesnminuitK is – it's the file in the kcorrs directory, but is it a file that is already supposed to be there, or is it the file that makesnminuitK is supposed to make? If so, a > is missing in the setup!* In principle, this should make a K-correction file in the "kcorrs" directory.
- The next step is to perform "S-corrections" with `colcorrectv4`. The parameters that it takes are: -r the ground light curve file (so none if one is only doing HST stuff), -h the HST light curve file (*this is the file in the hstlight directory – looks like it's the non-inverted version*), -gf the observed filter (e.g., r), -hf the HST equivalent filter (e.g., f675w), -rf1 the first rest filter (e.g., u), -rf2 the second rest filter if fitting two curves (e.g., b), -bm tmax (JD-2400000 of rest-B maximum), -s s, -col col (color at B-peak – ???), -z z, -host $E(B-V)_{\rm host}$, -mv $E(B-V)_{\rm MilkyWay}$, -hostrv host RV, -mwrv Milky Way RV, -cl a light curve file that will be created in the corrlight_noinv directory, -cli a light curve file that will be created in the corrlight_inv directory, -k1 the first k-correction file in the kcorrs directory (with the first set of rest-observed filters, such as "u" and "r"), -k2 the second k-correction file in the kcorrs directory if fitting two light curves at once. There are a number of other parameter that can be set in `colcorrectv4`, but these are the main ones.
- Then we do the actual fitting with SNMinuit. This invokes IDL, and runs SNMinuit with a .pro file constructed on the fly and put into the SCPfits directory. The parameters that SNMinuit needs are: the name of the supernova, one or two supernovae to fit, the rest and observed filters of the supernova(e), the lightcurve file(s) in corrlight_inv, and the k-correction file(s) in the kcorrs directory.

- The final step is to figure out the new host extinction or color. This is done with `evalextinc`, which also lives in the colv3 directory. The arguments for this are the two observed filters, col ??, dcol ??, z, stretch,$E(B-V)_{\rm MilkyWay}$, and host and Milky Way rv.

- At this point, we have a bunch of output files in the "SCPfits/Output_<first observed filter><second observed filter>" directory, which have the fitted supernova parameters, the χ^2, the errors, etc..

3. What `Simplehstltcv` does

`Simplehstltcv` starts with getting a user configuration. The user picks a background model, a charge-transfer-efficiency correction model, the number of pixels nx and ny that define the patch centered on the pixel given as the starting position for each NEW, and a psf. The idea here is that the SN is a point source, so the continuous image PSF represents the spread of the SN, on top of a structured continuous background already convoluted with the PSF (it is usually composed of the SN host galaxy, the halo of the surrounding objects, and a flat background of the sky light). Then it parses the input param file.

The next step is to set up the `HSTltcv` object.

- First, we set its background model (chosen above), and set the packground parameters to 0 if they are not in the input file.

- Then we set up the images.

  First, we let it know what its primary is, which is a `HSTltcvPatch`. We next add a vector of patches, which are of the type "patchinfo". Each `HSTltcv::addPatch` calls this patch's readPatch, with the following arguments: xctr, yctr (the position of the SN), nx, ny, and maybe theta if the image is rotated with respect to the primary. Then readPatches parses the ASCII part of the corresponding SCPImage (a .fits file), and figures out:

  - its epoch
  - detector temperature (needed for CTE corrections)
  - does edgesurfacing (??) to find out sky (FITSImage); average sky (sky -> Mean()); and error on the sky, dsky (original image -> sigma).

  Then we crop the original image, figure out how much data there is in the (nx x ny) square, and fill xofdata and yofdata. After that, we are done with addPatch, and we call the following for a given patch: setSigma to the error on the sky (dsky), the PSF model for this patch (ttDiff if nothing in the input file), and call setIndiv, which simply copies the indiv's (the files that went into making this NEW) exposure times and sky levels into dedicated vectors.

- After all the patches are thus added, we call `HSTltcv::endPatches`, which sets up the data covariance matrices (datacovar), with linear dimensions # patches * (nx x ny). The diagonal elements appear to be the background sigma (dsky^2) for a given patch + data (??), and the off-diagonal elements are all zero. This is also where we set up the order of patch parameters in the param array.

Then we do an LMFit and write result to the .psfresults fit The fitter takes the following input parameters:

- background model ones (6)
- flux for each patch (6) (these are read off the input file (NEW), the 1st parameter (what was previously called "starting magnitude"), and the units are counts/s)
- x,y for the patches with good transformations from the input file
- pedestals for patches from the input file

The fitter evaluates χ^2 in the following way. First, we loop over all the patches, and get data values at each point (call that y) – that's our xofdata and yofdata. Then, we evaluate what flux (with background) is expected for a given data point (this is hstltcv's evalFunc):

```
f = expTime *(background + flux*PSFval + pedestal),
```

where background = integral of the background model (use bkg parameters + x, y), PSF(use x,y), flux, and pedestral are obvious. This gives us a residual (y - f).

- The initial χ^2 = resid * M^{-1} * resid, here M is the covariance matrix that we calculated in endPatches.
- We then iterate using the following algorithm.
  - calculate df/da (derivatives of f above with respect to input parameters, which we will call a)
  - calculate α (that's 15.5.11)

$$\alpha_{kl} = \sum_{ij} MI^{ij} \left(\frac{df}{da}\right)_{ik} \left(\frac{df}{da}\right)_{jl} \tag{1}$$

    where MI is the inverse covariance matrix, k, l go over the parameter space (25), and i, j go over the data space

    Note that α_{kk} is multipiled by $(1 + \lambda)$, so it's really α'.

– calculate β

$$\beta_k = (1/2) \sum_{ij} \left(\frac{df}{da}\right)_{ik} MI^{ij}\, \mathrm{resid}_j \;+\; \mathrm{resid}_i\, MI^{ij} \left(\frac{df}{da}\right)_{jk} \tag{2}$$

– solve $\alpha * \Delta a = \beta$ for Δa, add Δa to the parameters, and evaluate χ^2 with these new parameters. If this new χ^2 is better, make λ bigger; otherwise, make it smaller.

– the whole process continues until $\Delta(\chi^2)$ becomes < 0.01, or the relative change in χ^2 becomes < 0.001.

- Output results: new parameters and covariance matrix (α).

4. Uber Spectrum

Many tools described above rely on the code in "UberSpec". For example, apart from parsing the input arguments, makesnminuitK does little other than call UberSpec's "kcorr" function. In this section, I'll describe what UberSpec's main functions do.

- **init(const string &filepath, const string &vegapath, const string &filterdir)** invokes the **readUber** spectrum at a given filepath (at the moment, it's in /home/users/rknop/deepsearch/data/snflux_1a.dat). The file is a table of dates, wavelengths, and flux densities (energy (or photons) per unit area per unit time per unit wavelength). Note that the flux density in the file is f_λ; however, it gets converted to n_λ by multiplying by λ in the code. It also reads in a file of flux densities (f_λ) for Vega in vegapath (currently the file is in /home/users/rknop/deepsearch/data/bessel/vega_kim.dat). The next step is to create vectors of filter names and filter paths (directories with filter information, based on filterdir).

- **getSpec(double myt, double z, double s, double hostebv, double mwebv, NLambda &spec)** adjusts the color based on the stretch. First, it invokes the "effective time" efft and finds the spectrum at this time:

$$\mathrm{efft} = \frac{\mathrm{myt}}{s\,(1.+z)} \tag{3}$$

why divide by stretch? If s is within 0.005 of 1, the assumption is that the spectrum has all the same colors as an $s = 1$ supernova. If not, the following steps are taken:

– if $E(B-V)_{\mathrm{host}}$ is non-trivial, then the spectrum is reddened by invoking its **redden(double ebv, double RV)** method. This method uses the extinction law from (1).

  – if z is non-trivial, then the spectrum is redshifted by invoking its **redshift(z)** method. This method multiplies the λ by $(1+z)$, and divides n_λ by $(1+z)^2$.
  – if $E(B-V)_{\rm MilkyWay}$ is non-trivial, then the spectrum is reddened.

  The final spectrum is returned in spec.

- **clr(const string &filter1, const string &filter2, double t, double z, double s, double hostebv=0., double mwebv=0., double zfilt1=0.)** does the following. First, the first (rest) filter is redshifted if requested (if zfilt1 is non-zero – this is used for K-corrections). Then, getSpec is invoked with the arguments t, z, s, hostebv, and mwebv. The returned spectrum is "cropped" (so that its wavelengths are covered by either filter1 (let's call this spectrum spec1) or filter2 (spec2)), multiplied by either filter1 or filter2, and then integrated over their respective wavelength ranges:

$$\mathrm{clr} = -2.5\log 10\left[\frac{\int d\lambda_1 n_\lambda[(1+z)\lambda_1]}{\int d\lambda_2 n_\lambda(\lambda_2)}\,\frac{\int d\lambda_2 n_\lambda^{\rm Vega}(\lambda_2)}{\int d\lambda_1 n_\lambda^{\rm Vega}(\lambda_1)}\right] \tag{4}$$

  where λ_1 is the rest frame wavelength (so it is redshifted if requested), observed through filter 1; λ_2 is the observed wavelength, observed through filter 2.

- **kcorr(const string &filter1, const string &filter2, double t, double z, double s, double hostebv=0., double mwebv=0.)** invokes the clr function with the last argument (zfilt1) set to 1. It is

$$\mathrm{kcorr} = -2.5\log 10(1+z)\; - \;\mathrm{clr} \tag{5}$$

- **mwExtCorr(const string &obsfilt, double t,double z,double s, double hostebv,double mwebv)** is used for figuring out extinction corrections for Milky Way. This is done by invoking getSpec with mwebv set to 0 (let's call the returned spectrum spec1), redenning spec1 (let's call that spec2), and cropping both spectra by obsfilt. The final correction is

$$\mathrm{mwextcorr} = -2.5\mathrm{log}10\frac{\int spec1}{\int spec2} \tag{6}$$

5. Color Corrections

The following steps are taken in `colcorrectv4` to color-correct light curves.

- The default way of getting the color corrections in `colcorrectv4` is through interpolating templates. We create templates using `LTCVTemplate`, with the following arguments: z, s, restfilter (e.g., u), observed filter (e.g., r), and the corresponding K-correction file. Creating such a template basically amounts to reading the master_template.dat file (in the colv3 directory), which is a table of supernova magnitudes in various rest filters vs. day.

- Then we calculate the color, which is the difference in K-corrected magnitudes for both sets of filters plus the peakcolor (??), for each epoch. The error on the color seems to be hard-coded and set to 0.03 for all epochs (??).

- The next step involves performing star-supernova corrections. This is basically performing color-corrections based on direct UberSpectru integration, *not* using linear or quadratic color terms. It may be relative to a star of the same color. This step is done by invoking **LTCV::uberColCorrect** for the HST spectrum. I am not very clear on why it's done the way it is, but here's what's done. For each epoch, we get a supernova spectrum by using UberSpec's getSpec. Let's call this spectrum sn. We then integrate it, as well as Vega's spectrum over the filter in in /home/users/rknop/deepsearch/data/bessel/sr.dat. Let's call -2.5 log10 of the ratio between the result of the integration of sn and vega sn1. We then do the same thing using the filter in /home/users/rknop/deepsearch/data/bessel/si.dat, obtaining sn2. The same thing using the filter in /home/users/rknop/deepsearch/data/hst/r1.dat results in snhst; and using /home/users/rknop/deepsearch/data/bessel/sr.dat, in snground. We call the difference between snground and snhst a supernova color correction for a given epoch. The uncertainties in the corrections are assumed to be 10% of the corrections, and 100% correlated. The corrections (which are magnitudes!) are then converted to flux and applied to both the original fluxes and the covariance matrix.

- The next step is merging the ground and the hst light curves. Even if there is no ground light curve, the hst light curve is merged with a "junk" one. The reason this is done is so that the zero-point errors are forced into the covariance matrix.

- The final step involves writing out the corrected light curves into the corrlight_inv and corrlight_noinv directories.

REFERENCES

J. O'Donnell, ApJ **422**, 158 (1994)