kaepora Documentation

Release 1.0

Matt Siebert

Oct 03, 2022

Contents

kaepora is an open-source relational database of Type Ia Supernova observations. This guide is meant to outline how to install and interact with the database. We also provide tools for creating composite spectra using the methods from Siebert et al. 2019.

Contents:

It appears that the time has finally come for you to start your adventure! You will encounter many hardships ahead... That is your fate. Don't feel discouraged, even during the toughest times!

CHAPTER 1

Prerequisites

Python 2.7

numpy

matplotlib

sqlite3

scipy

astropy

specutils

tabulate

Version specific dependencies:

msgpack-python version 0.4.6

msgpack-numpy version 0.3.6

CHAPTER 2

Downloading/Building the Database

The source code for *kaepora* can be found at https://github.com/msiebert1/kaepora. First clone the repository using:

```
git clone https://github.com/msiebert1/kaepora.git
```

We recommend that you download the most recent version of the database from https://msiebert1.github.io/kaepora/. Unzip the folder and place the '.db' file in the /data folder of your repository.

Alternatively, you can build the database from source. This process runs several scripts that homogenize the raw spectral data and takes several hours. If you wish to do this navigate to the /kaepora/src folder and execute the following command:

```
python build_kaepora.py
```

Once you have completed one of these steps you should be ready to interact with the database.

CHAPTER 3

Querying the Database

The database currently consists of two tables: *Spectra* and *Events*. These tables host the spectrum-specific and SN-specific metadata respectively. Currently the only way to interact with the database is via an SQL join on these tables. Right now, you should run your code from the /src directory. Our documentation will focus on the routines available in the `kaepora.py`. Start by importing this module:

```
import kaepora as kpora
```

You can then define an array containing SQL queries and obtain spectra from the database. For example:

```
example_query = ["SELECT * from Spectra inner join Events ON Spectra.SN = Events.SN␣
↪where phase >= -1 and phase <= 1 and ((dm15_source < 1.8) or (dm15_from_fits < 1.8))
↪"]
spec_array = kpora.grab(example_query[0])
```

If you would like to remove atypical SNe Ia, SNe with flagged artifacts, and SNe with poor host reddening corrections use:

```
spec_array = kpora.grab(example_query[0], make_corr=True)
```

These spectra have been already been corrected for MW reddening. To correct these spectra for host-galaxy reddening (and exclude SNe with $A_V > 2$) with a F99 reddening law use:

```
spec_array = kpora.host_dereddening(spec_array, cutoff=2.)
```

CHAPTER 4

Spectrum Objects

`spec_array` now contains an array of objects that contain our homogenized spectra and all of the spectrum- and SN-specific metadata. Currently these objects are made to represent single spectra, so objects generated from the same SNe will contain some redundant SN metadata. These spectra are normalized to their maximum flux. Basic information on these objects can be viewed with:

```
for spec in spec_array_dered:
    print spec.name, spec.filename, spec.source, spec.phase, spec.wavelength[spec.x1],
↪ spec.wavelength[spec.x2]
```

A spectrum and its variance can be plotted with:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(2,1)
example_spec = spec_array_dered[20]
ax[0].plot(example_spec.wavelength, example_spec.flux)
ax[1].plot(example_spec.wavelength, 1/example_spec.ivar)
plt.show()
```

Below we describe other attributes of these objects that are also queryable parameters of the database.

CHAPTER 5

Schema

5.1 Spectral Attributes

Attribute	SQL Format	Description	Type
name	"SN"	SN name	String
filename	"Filename"	Filename from data source	String
source	"Source"	Data source	String
minwave	"Minwave"	Minimum wavelength of original spectrum	float
maxwave	"Maxwave"	Maximum wavelength of original spectrum	float
SNR	"snr"	Median S/N of the spectrum	float
mjd	"MJD"	Modified Julian Date of the spectrum	float
phase	"Phase"	Rest-frame days from B-Band maximum	float
ref	"Ref"	Bibtex code	String

5.2 SN Attributes

These attributes contain the most metadata. We also include (but do not list) metadata from the results of several different light curve fits. If you would like to construct a query based on these metadata please contact me.

Attribute	SQL Format	Description	Type
redshift	"Redshift"	Redshift from NED	float
mjd_max	"MJD_max"	Modified Julian date corresponding to the time of maximum-light	float
dm15_source	"Dm15_source"	Dm15 from the source survey	float
dm15_from_fits	"Dm15_from_fits"	Dm15 calculated from the polynomial relationship with a light-curve shape parameter	float
e_dm15	"e_dm15"	Error in dm15	float
av_25	"Av_25"	Estimated host galaxy extinction from an MLCS fit using R_v = 2.5	float
m_b_cfa	"M_b_cfa"	Absolute B-Band magnitude at maximum light from the CfA sample	float
m_b_cfa_err	"M_b_cfa_err"	Error in m_b_cfa	float
b_minus_v_cfa	"B_minus_V_cfa"	B-V color at maximum light	float
b_minus_v_cfa_err	"B_minus_V_cfa_err"	Error in b_minus_v_cfa	float
v_at_max	"V_at_max"	Estimated velocity at maximum light	float
v_err	"V_err"	Error in v_at_max	float
ned_host	"NED_host"	A simplified version of NED host galaxy morphology based on cross-listed objects	String
carbon	"Carbon_presence"	'A': carbon detected, 'F'; marginal carbon detected, 'N': no carbon detected	String
hubble_res	"Hubble_res"	Hubble residual from a SALT fit	float

You can view all attributes of the spectrum object with the code below:

```
spec_attributes = dir(spec_array[0])
print spec_attributes
```

CHAPTER 6

Creating Composite Spectra

Here we outline how to generate composite spectra using the methods of Siebert et al. 2019. Start by defining `query_list` such that it describes the subset of data that for which you wish to generate a composite spectrum. Then run the `make_composite`:

```
query_list = ["SELECT * from Spectra inner join Events ON Spectra.SN = Events.SN␣
↪where phase >= -1 and phase <= 1 and ((dm15_source < 1.8) or (dm15_from_fits < 1.8))
↪"]
composites, sn_arrays, boot_sn_arrays = kpora.make_composite(query_list, boot=False,␣
↪medmean=1, verbose=False, gini_balance=True, combine=True)
```

This will generate a composite spectrum for each query in `query_list` and will take of order seconds to minutes depending on the size of the sample. A composite spectrum is a spectrum object that contains a few more attributes. `phase_array`, `dm15_array`, and `red_array` contain the weighted averages of phase, dm15, and redshift as a function of wavelength. `sn_arrays` contains the list of combined spectrum objects used to construct the composite spectrum.

Other applications of the available `make_composite` arguments are listed below

`boot = True`: estimate the 1-sigma confidence intervals via 100 random resamples of the spectra (this significantly increases computation time). This will populate the `low_conf` and `up_conf` attributes of the composites spectrum objects. This also populates `boot_sn_arrays` with a list of composite spectra generated from these resamples.

`medmean = 2`: a median composite spectrum will be generated.

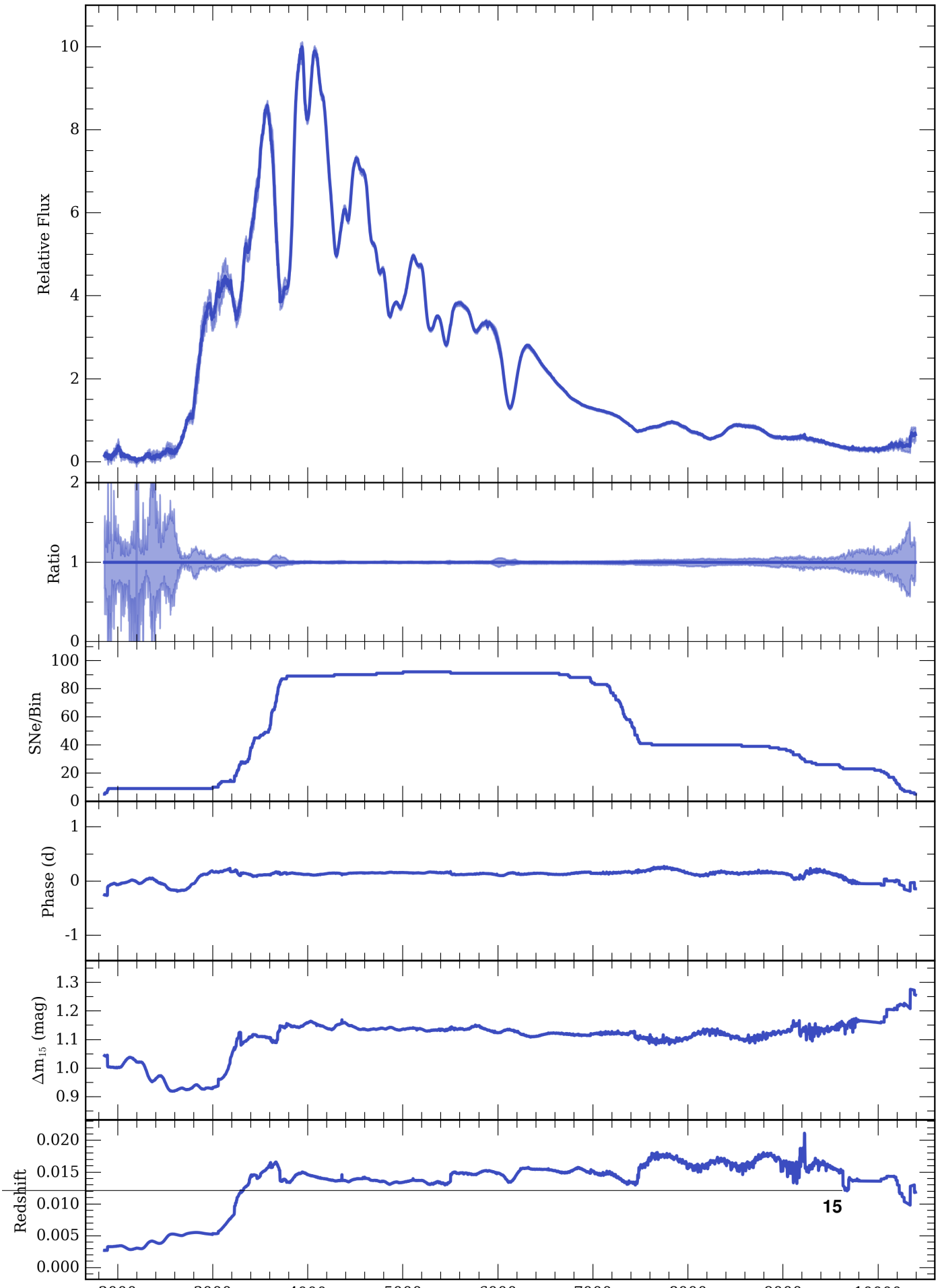
`verbose = True`: prints basic information about each spectrum contributing to the composite spectrum.

`gini_balance = False`: does an inverse-variance weighted average of the original data (this is more susceptible to high SNR outliers). For more information on our Gini-weighting method please read our paper.

`combine = False`: does not initially combine spectra from the same SNe.

We also provide a useful plotting function to visualize your composite spectra. This will also output the average properties of the composite spectra within wavelength ranges defined by `set_min_num_spec`.

```
import kaepora_plot as kplot
kpora.set_min_num_spec(composites, 5)
kplot.comparison_plot(composites, min_num_show=5)
```


Relative Flux
Ratio
SNe/Bin
Phase (d)
Δm_{15} (mag)
Redshift

CHAPTER 7

Indices and tables

- genindex
- search