SALT2 Training Documentation

David Jones, Rick Kessler

Jun 06, 2023

CONTENTS

CHAPTER

ONE

OVERVIEW

SALT is a model of Type Ia supernovae (SNe Ia) that accounts for spectral variations as a function of shape and color (Guy et al., 2007; Guy et al., 2010; Betoule et al., 2014). With SALTShaker we have developed an open-source model training framework and created the "SALT3" model. We more than doubled the amount of photometric and spectroscopic data used for model training and have extended the SALT framework to 11,000 Angstroms. In the coming years, SALT3 will make use of data from the Vera Rubin Observatory, and the *Nancy Grace Roman Space Telescope* and can be re-trained easily in the coming years as more SN Ia data become available.

Please report bugs, issues and requests via the SALTShaker GitHub page.

CHAPTER

TWO

SALT3 MODEL AND TRAINING DATA

The latest version of the SALT3 model has been released in:

Taylor et al., 2023, MNRAS, 520, 5209T

This model includes full re-calibration of the SALT3 training data (Brout et al., 2021) to match SALT training sets used in the Pantheon+ analysis. Other SALT3 publications include:

- Pierel et al., 2021, ApJ, 911, 96P: model-independent simulation framework for SALT3 validation.
- Kenworthy et al., 2021, ApJ, 923, 265K: SALT3 model and SALTShaker framework.
- Pierel et al., 2022, ApJ, 939, 11P: A near-infrared extension to the SALT3 model.
- Dai et al., 2023, ApJ, in press: SALT3 model validation with extensive simulations and full training pipeline.
- Jones et al., 2023, ApJ, in press: a host-galaxy mass-dependent SALT3 model.

The latest SALT3 model files are linked here. SALT3 light curve fits can be performed using sncosmo (currently the latest version on GitHub is required) or SNANA with the SALT3.K21 model, with a brief sncosmo example given below.

The latest SALT3 training data is also fully public and included here. This release includes all photometry and spectra along with everything required to run the code. Once SALTShaker has been installed via the instructions in *Installation*, the SALT3 model can be (re)trained following the instructions in *Getting Started Quickly*.

CHAPTER

THREE

EXAMPLE SALT3 FIT

Fitting SN Ia data with SALT3 can be done through the sncosmo or SNANA software packages. With sncosmo, the fitting can be performed in nearly the exact same way as SALT2. Here is the example from the sncosmo documentation, altered to use the SALT3 model. First, install the latest version of sncosmo; SALT3 is included beginning in version 2.5.0:

```
conda install -c conda-forge sncosmo
```

or:

```
pip install sncosmo
```

Then, in a python terminal:

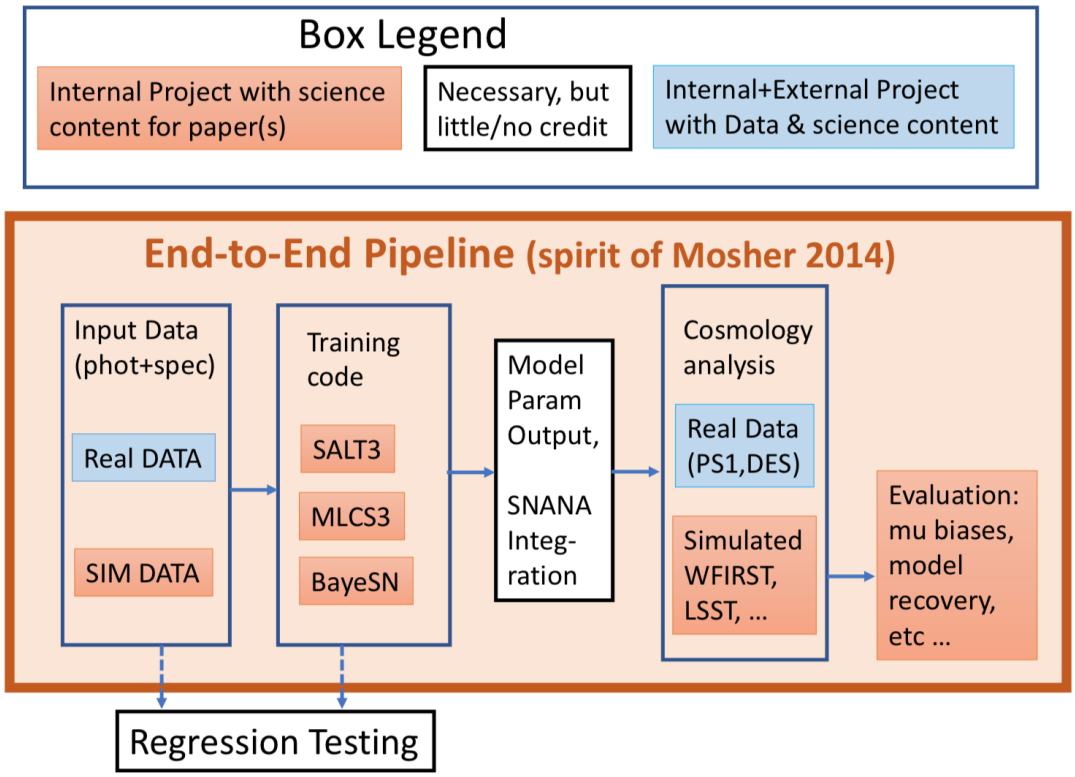
```
import sncosmo
data = sncosmo.load_example_data()
model = sncosmo.Model(source='salt3')
res, fitted_model = sncosmo.fit_lc(data, model,
                                   ['z', 't0', 'x0', 'x1', 'c'],
                                   bounds={'z':(0.3, 0.7)})
sncosmo.plot_lc(data, model=fitted_model, errors=res.errors)
```

CHAPTER

FOUR

PIPELINE

In Dai et al., 2023 we present a pipeline to fully test and validate the SALT3 model in the context of cosmological measurements. Defails are given in *Running the Pipeline*.



4.1 Installation

4.1.1 Install from PyPI

SALTShaker can be installed with PyPI, but jax might need to be installed separately via conda first:

```
conda create -n saltshaker python=3.10
conda activate saltshaker
conda install -c conda-forge jax
pip install saltshaker-sn
```

Check out *Getting Started Quickly* to start using SALTShaker.

4.1.2 Install from GitHub

To install via GitHub:

```
git clone https://github.com/djones1040/SALTShaker.git
```

If you wish, create an isolated conda environment for the code:

```
conda create -n saltshaker python=3.10
conda activate saltshaker
conda install -c conda-forge jax
```

Finally, install the code with:

```
cd SALTShaker
pip install .
```

4.2 Getting Started Quickly

To make it easy to run the latest SALTShaker training, the *trainsalt -g* command downloads and unpacks the training data/config files into a local directory called *saltshaker-latest-training*. You can run the full training (takes a few hours) with the following commands:

```
conda activate saltshaker
trainsalt -g
cd saltshaker-latest-training
trainsalt -c traingradient.conf
```

Training results will be in the *output/* directory.

4.3 Spectroscopic and Photometric Training Data

SALTShaker input files use SNANA format, which allows easy synergy between model training and SN simulations, light-curve fitting, and systematic uncertainty estimation. The SNANA-formatted data necessary for training includes photometry, spectroscopy, and filter functions/photometric system information.

For photometry and spectroscopy, a number of light curves and spectra are provided in the `examples/SALT3TRAIN_K21_PUBLIC/` directory for training. Light curves and spectra are combined into a single file. The training data themselves are described in Kenworthy et al., 2021.

For the photometric information, so-called "kcor" files - which confusingly contain no k-corrections - are given in the `examples/SALT3TRAIN_K21_PUBLIC/kcor` directory. These FITS-formatted files define the photometric system associated with each survey that comprises the training sample. The SNANA function `kcor.exe` will create these files from the `.input` files in the same directory if anything needs to be adjusted. "kcor" files contain filter transmission functions, AB, BD17, or Vega spectra depending on the photometric system of the data, zeropoint offsets, and optional shifts to the central wavelength of each filter.

4.3.1 Photometry and Spectroscopy Format

SNANA file format consists of a number of header keys giving information about each SN, followed by photometry and spectroscopy.

An example of the minimum required header is below:

```
SURVEY: FOUNDATION
SNID: ASASSN-15bc
RA: 61.5609874
DEC: -8.8856098
MWEBV: 0.037 # Schlafly & Finkbeiner MW E(B-V)
```

Below the header, the photometry is included in the following format:

```
NOBS: 64
NVAR:   7
VARLIST:  MJD  FLT FIELD   FLUXCAL   FLUXCALERR    MAG     MAGERR
OBS: 57422.54 g VOID  21576.285 214.793 16.665 0.011
OBS: 57428.47 g VOID  30454.989 229.733 16.291 0.008
OBS: 57436.55 g VOID  26053.054 253.839 16.460 0.011
OBS: 57449.46 g VOID  11357.888 158.107 17.362 0.015
...
END_PHOTOMETRY:
```

The SALT3 training code only reads the MJD, FLT (filter), FLUXCAL, and FLUXCALERR values. FLUXCAL and FLUXCALERR use a zeropoint of 27.5 mag.

The beginning of the spectroscopy section is identified by the following header lines:

```
NVAR_SPEC: 3
VARNAMES_SPEC: LAMAVG  FLAM  FLAMERR
```

Where the columns are wavelength (angstrom), flux (erg/cm^2/s/A), and flux uncertainty (not currently used). Each spectrum has the following format:

```
SPECTRUM_ID:       1
SPECTRUM_MJD:      54998.378  # Tobs =  -13.832
SPECTRUM_TEXPOSE:  100000.000  # seconds
SPECTRUM_NLAM:     352 (of 352)  # Number of valid wavelength bins
SPEC:  4200.00  4209.35  -2.068e-10   5.701e-10
SPEC:  4209.35  4218.76  -2.704e-10   6.359e-10    2.557e-10  23.25
SPEC:  4218.76  4228.23  -2.725e-10   6.312e-10    2.543e-10  23.26
SPEC:  4228.23  4237.76  -4.588e-11   6.232e-10    2.538e-10  23.25
SPEC:  4237.76  4247.35  -8.320e-10   6.152e-10    2.541e-10  23.25
...
END_SPECTRUM:
```

4.4 Training the SALT3 Model

usage:

```
trainsalt -c <configfile> <options>
```

Although there are a number of training configuration files in the `examples/` directory, the simplest way to train the SALT3.K21 model with all data and spectra and with the latest calibrations is to use the configuration files and data in the `examples/SALT3TRAIN_K21_PUBLIC` directory.

To train the SALT3.K21 model, run:

```
trainsalt -c Train_SALT3_public.conf
```

This directory contains all the lightcurves, spectra, and filter definition files needed to train the model, with outputs in the `output` directory.

The training is slow given the large data volume and takes approximately 1 to 1.5 days, but can be sped up with a couple reasonable choices. The first is changing the `steps_between_errorfit` argument to estimate model uncertainties less frequently, as uncertainty estimation (~4.5 hours) is the slowest component of the code:

```
trainsalt -c Train_SALT3_public.conf --steps_between_errorfit 15
```

Another option is to bin the spectra, which will reduce the amount of spectroscopic data points by an order of magnitude:

```
trainsalt -c Train_SALT3_public.conf --binspec True
```

This should not result in any noticeable difference to the model surfaces but hasn't yet been tested fully. Additional speed and memory usage improvements are currently in progress.

4.4.1 SALT3 Training Configuration Options

See the `examples/SALT3TRAIN_K21_PUBLIC/Train_SALT3_public.conf` file and the `examples/SALT3TRAIN_K21_PUBLIC/training.conf` files for the full list of training options. Two configuration files are used with the goal that users should rarely have to modify the default `training.conf` options. Descriptions of each option are given below.

Name	Default	Description
main config file		
[iodata]		
snlists		ASCII file or comma-separated list of files. Each file contains a list of SN files (
tmaxlist		Time of maximum light for each SN in training. See `examples/SALT3TRAIN_`
snparlist		initial list x0,x1,c and FITPROB (prob. that the data matches model, from SAL
specrecallist		Option to provide an initial set of spectral recalibration parameters. No longer r
dospec	True	If set, use spectra in training
maxsn	None	Debug option to limit the training to a given number of SNe
outputdir		Directory for trained model outputs
keeponlyspec	False	Debug option - keep only those SNe with spectroscopic data
initm0modelfile	Hsiao07.dat	Initial SN SED model. Initial parameter guesses are derived from this file. Defa
initm1modelfile		Initial SN SED model. Will guess M1 from a time-dilated Hsiao model if no fil
initsalt2model	True	If True, use SALT2 as the initial guess. Otherwise use initm0modelfile.
initsalt2var	False	If set, initialize model uncertainties using SALT2 values. No longer recommend
initbfilt	Bessell90_B.dat	Nominal *B*-filter for putting priors on the normalization

Table 1 – continued from previous page

Name	Default	Description
resume_from_outputdir		Resume the training from an existing output directory
resume_from_gnhistory		If resume_from_outputdir is set, set to same directory name to resume training f
loggingconfig	logging.yaml	Gives configuration options for the training logs
trainingconfig	training.conf	Additional configuration file. Will look in the package directory if it's not found
calibrationshiftfile		A file that can adjust the calibration of the input files, e.g. for estimating system
filter_mass_tolerance	0.01	Amount of filter "mass" allowed to be outside the SALT wavelength range
fix_salt2modelpars	False	Debug option - if True, does not fit for M0 and M1.
validate_modelonly	False	If True, only produces model validation plots but not plots spectra or lightcurves
[survey_<SURVEY>]		The parameters file requires a category for **every** SURVEY key in SN data files
kcorfile		Kcorfile (includes filter ZPT offsets and filter definitions) for each SURVEY key i
subsurveylist		Comma-separated list of sub-surveys for every survey, e.g. CFA4 is the subsurve
[trainparams]		
gaussnewton_maxiter	30	Maximum number of Gauss-Newton iterations allowed if convergence (delta ch
regularize	True	Include regularization if True
fitsalt2	False	Try to fit SN parameters with SALT2 model in the validation stage if True
n_repeat	1	**deprecated, leave alone**
fit_model_err	True	If True, fits model errors every steps_between_errorfit iterations
fit_cdisp_only	False	If True and fit_model_err is True, fits for the color scatter but no other mode
steps_between_errorfit	5	Estimate model errors every x iterations
model_err_max_chisq	4	Begin estimating model errors when the reduced chi^2 of the training is below t
condition_number	1e-80	Conditioning matrices for the Gauss-Newton process. Leave this alone.
fit_tpkoff	False	if true, fit for time of maximum light along with other parameters (not well teste
fitting_sequence	all	optionally, can fit for different model components in sequence. Can make it harc
training.conf file		**In most cases, leave these alone**
[trainingparams]		
specrecal	1	if 1 (or True), do the spectral recalibration
n_processes	1	**deprecated**
estimate_tpk	False	**not recommended** estimate time of maximum light for each SN before beginni
fix_t0	False	**deprecated**
n_min_specrecal	3	minimum number of parameters for the spectral recalibration polynomial
n_max_specrecal	10	maximum number of parameters for the spectral recalibration polynomial
regulargradientphase	1e4	amplitude of gradient regularization chi^2 penalty for phase (**semi-arbitrary**)
regulargradientwave	1e5	amplitude of gradient regularization chi^2 penalty for wavelength (**semi-arbitra**
regulardyad	1e4	amplitude of dyadic regularization chi^2 penalty (**semi-arbitrary**)
m1regularization	100	multiply regularization amplitude for the M1 component by this amount (**semi-a**
specrange_wavescale_specrecal	2500	normalizes the spectra for recalibration
n_specrecal_per_lightcurve	0.5	add one spectral recal parameter for every two photometric bands in a given SN
regularizationScaleMethod	fixed	options for adjusting regularization scale in training/saltresids.py
wavesmoothingneff	1	Gaussian smoothing scale for the amount of training data at each wavelength for
phasesmoothingneff	3	Gaussian smoothing scale for the amount of training data at each phase for smoo
neffloor	1e-4	below neffloor, regularization does not continue to increase in strength
neffmax	0.01	above neffmax, regularization is turned off
binspec	False	use spectral binning if True
binspecres	29	resolution of the spectral binning
spec_chi2_scaling	0.5	tuned so that spectra and photometry contribute ~equally to total chi^2 in trainin
[modelparams]		
waverange	2000,11000	wavelength range over which the model is defined
colorwaverange	2800,8000	wavelength range over which the color law polynomial is fit
interpfunc	bspline	function for interpolating the model between control points (b-spline is default)

Table 1 – continued from previous page

Name	Default	Description
errinterporder	0	order of the spline interpolation for the errors
interporder	3	order of the spline interpolation for the model
wavesplineres	69.3	number of Angstroms between wavelength control points
waveinterpres	10	wavelength resolution of the model used during training (Angstroms)
waveoutres	10	wavelength resolution of the trained model written to output directory (Angstro
phaserange	-20,50	phase range over which the model is defined (rest-frame days)
phasesplineres	3.0	phase resolution of the trained output model (days)
phaseinterpres	0.2	phase resolution of the model used during training (days)
phaseoutres	1	phase resolution of the trained model written to output directory (days)
n_colorpars	5	number of parameters used to define the color law polynomial
n_colorscatpars	5	number of parameters used to define the color scatter
n_components	2	number of model components (M0, M1) - additional components not yet allowe
error_snake_phase_binsize	6	spacing in days for the SALT error model B-spline interpolation
error_snake_wave_binsize	1200	spacing in Angstroms for the SALT error model B-spline interpolation
use_snpca_knots	False	if true, use the knot locations from the SALT2 training
[priors]		key is the name of a decorator in `training/priors.py`; value determines the
x1mean	0.1	mean x1 = 0
x1std	0.1	standard deviation of x1 values = 1
m0endalllam	1e-5	at -20 days, M0 must go to zero flux
m1endalllam	1e-4	at -20 days, M1 must go to zero flux
colorstretchcorr	1e-4	color and stretch should not be correlated
colormean	1e-3	mean sample color is zero
m0positiveprior	1e-2	M0 is not allowed to be negative
recalprior	50	don't allow spectral recalibration to go crazy
[bounds]		
x1	-5,5,0.01	min,max,prior width on x1

4.5 Flexible Simulated Data with BYOSED

4.5.1 "Build Your Own" SED

The BYOSED framework allows any spectrophotometric model to be used as the underlying template to generate simulated Type Ia light curve data with SNANA. This framework is published in Pierel et al., 2021. By default, this model is the Hsiao+07 model (initfiles/Hsiao07.dat). This can be replaced by any model.

4.5.2 Param File Basics

The only file to set up is the BYOSED.params file. This contains the general aspects of the simulated SN you want to create using BYOSED, and any warping effects you want to add in. This file is separated into the following required and optional sections:

[MAIN]

(Required)

This section contains **SED_FILE** (name of SED file), as well as **MAGSMEAR** (magnitude smearing) and **MAGOFF** (magnitude offsets) definitions to be applied to the base SED defined by sed_file. You may also define **CLOBBER** and **VERBOSE** flags here as well. This section may look like the following:

```
[MAIN]

SED_FILE: Hsiao07.dat
MAGSMEAR: 0.0
MAGOFF: 0.0
```

[FLAGS]

(Optional)

This section allows you to simply turn warping effects defined in the next section(s) on and off. If this section exists, then it supersedes later sections and defines the warping effects to be used. If it does not exist, all defined warping effects are used. Adding this onto the **[MAIN]** section, the params file might now look like the following:

```
[MAIN]

SED_FILE: Hsiao07.dat
MAGSMEAR: 0.0
MAGOFF: 0.0

[FLAGS]

COLOR: True
STRETCH: True
HOST_MASS: False
```

In this case, a magnitude smearing of 0.1 would be applied to the Hsiao model at all wavelengths, and some color and stretch effects are applied as well based on functions you will define in the next sections.

4.5.3 Warping Effects

The following sections contain all of the various wavelength/phase dependent effects that you want to apply to your SED. In this case, based on the **[FLAGS]** section, you must have a "COLOR" section and a "STRETCH" section. You can name effects whatever you want **with the exception of a "color law" effect, which must be named **"COLOR"**, as long as the name of your section and the corresponding name in the **[FLAGS]** section are identical. Creating a warping effect section requires the following variables in no particular order:

1. DIST_PEAK
   - The PEAK of an (a)symmetric Gaussian that will define the distribution for the scale parameter
2. DIST_SIGMA
   - The "low" and "high" standard deviations of the same distribution
3. DIST_LIMITS
   - The lower and upper cutoff you would like for the same distribution
6. DIST_FUNCTION
   - A file name to be read that contains a list of phase, wave, value like the following:

```
#p w v
-20 1000 25.75805
-20 1010 25.64852
-20 1020 25.53899
-20 1030 25.42946
-20 1040 25.31993
-20 1050 25.2104
     ...
```

You must now define a section for each warping effect, with these variables. For our current example, where I have defined color and stretch effects in my **[FLAGS]** section, I must define these two sections. If I do not define a **[FLAGS]** section, then whatever sections that exist apart from the **[MAIN]** section are assumed to be warping effects. One such section might look like the following:

```
[COLOR]

WARP_FUNCTION: color_func.dat
DIST_PEAK: 0.0
DIST_SIGMA: 0.07 0.1
DIST_LIMITS: -0.3 0.3
```

All together, after adding in the stretch section as well, a **BYOSED.params** file might look something like this:

```
[MAIN]

SED_FILE: Hsiao07.dat
MAGSMEAR: 0.0
MAGOFF: 0.0

[FLAGS]

COLOR: True
STRETCH: True
HOST_MASS: False
```

(continues on next page)

(continued from previous page)

```
[COLOR]

WARP_FUNCTION: color_func.dat
DIST_PEAK: 0.0
DIST_SIGMA: 0.07 0.1
DIST_LIMITS: -0.3 0.3

[STRETCH]

WARP_FUNCTION: stretch_func.dat
DIST_PEAK: 0.5
DIST_SIGMA: 1.0 0.7
DIST_LIMITS: -2.5 2.5
```

Or, if you do not define a flags section, color and stretch will automatically be used as warping effects with the following **BYOSED.params** file:

```
[MAIN]

SED_FILE: Hsiao07.dat
MAGSMEAR: 0.0
MAGOFF: 0.0

[COLOR]

WARP_FUNCTION: color_func.dat
DIST_PEAK: 0.0
DIST_SIGMA: 0.07 0.1
DIST_LIMITS: -0.3 0.3

[STRETCH]

WARP_FUNCTION: stretch_func.dat
DIST_PEAK: 0.5
DIST_SIGMA: 1.0 0.7
DIST_LIMITS: -2.5 2.5
```

4.5.4 Final Notes

Now you can replace the Hsiao template with your own template SED, and start adding in warping effects. This warping process is designed so that as many effects as you would like can be included. Anything but a color effect (which should affect the final SED as a function of wavelength and possibly phase) is applied additively, while the color effect is applied multiplicatively. This is similar to the existing SALT2 framework. For the example file above, the final flux would look like this

$$F(\lambda,\phi) = A\Big[H(\lambda,\phi) + S(\lambda,\phi)s\Big] \times 10^{-0.4C(\lambda,\phi)c}$$

Where here F is the final flux, H is the Hsiao template, S is the defined stretch function, C is the defined color function, s is the scale parameter pulled from the distribution defined for the stretch function, and c is the scale parameter pulled from the distribution defined for the color function. In principle this could look like the following if you had N such

effects:

$$F(\lambda,\phi) = A\Big[H(\lambda,\phi) + X_1(\lambda,\phi)x_1 + X_2(\lambda,\phi)x_2 + ... + X_N(\lambda,\phi)x_N\Big] \times 10^{-0.4C(\lambda,\phi)c}$$

4.5.5 Example Files

These are example files that can be used for your `sed_file` and `BYOSED.params`. The color and stretch functions are defined by accompanying `color` and `stretch` files.

4.6 Running the Pipeline

4.6.1 Pipeline Discription

The SALT3 Training pipeline consists of several procedures that will be run in series. The pipeline modifies a base input file to create a customized one and calls the external program with the customized input. Details are described below and the pipeline will be described and published in Dai et al. (in prep.).

4.6.2 Param File

General Structure

Each section in the param file defines one procedure in the pipeline. The gerenal structure is as follows:

```
[Procedure Name] ([byosed], [simulation], [training], [lcfitting], [cosmology], ...)

# external program to call
pro =

# arguments for the external program
proargs =

# base input location
baseinput =

# define the section (optional), key and value to be added or changed from the base input

set_key= [NCOL] # 2 if no section or 3 if section exists in the config file
    [SECTION1] [KEY] [VALUE]
    [SECTION2] [KEY2] [VALUE2]
    [SECTION2] [KEY3] [VALUE3]
```

Batch mode

The pipeline supports batch submission for certain stages (e.g. simulation, lcfitting, . . .)

set *batch=True* under that stage.

4.6.3 Running the Pipeline

The pipeline class

The pipeline can be run using the SALT3pipe class. In the `examples/pipelinetest` directory, you can run the pipeline with the following commands:

```
from salt3.pipeline.pipeline import *
pipe = SALT3pipe(finput='sampleinput.txt')
pipe.build()
pipe.configure()
pipe.run()
```

Building the pipeline with selected stages

The `build` method need to be called before `configure` and `run` The default pipeline includes all the stages. Currently they are `['byosed','sim','train','lcfit','getmu','cosmofit']` or `['data','train','lcfit','getmu','cosmofit']`, depending on the value of the `data` option. This can be set simply by

```
pipe.build()
```

The option `data` can be turned on/off to use data/sims, for example:

```
pipe.build(data=False)
```

The default value is `data=True`

To specify or skip certain stages, set the option `mode='customize'`, and specify/skip stages using `onlyrun/skip`. Note that the only one of the options can be set.

```
pipe.build(data=False,mode='customize',onlyrun=['lcfit','getmu','cosmofit'])
```

Once the `build` method is called, the `configure` method need to be called following it so that the input files are properly configured.

Connecting the input/output of different stages using the 'glue' method

The `glue` method can be called so that the input and output of the gluing stages are properly connected. This will overwrite the config (input) files of the stages and should be called after `configure`.

```
pipe.glue(['sim','train'])
```

For some stages that are connected with multiple stages, the `on` option specify what input/output files to glue on:

```
pipe.glue(['train','lcfit'],on='model')
```

```
pipe.glue(['sim','lcfit'],on='phot')
```

Running the pipeline

After calling `build` and `glue`, call the `run` method the execute the pipeline:

```
pipe.run()
```

Note the `build`, `configure`, `glue` and `run` methods can be called multiple times to build a customized pipeline. Keep in mind each time `configure` is called, it modifies the config (input) file of certain stages in specified in `build`; and each time `glue` is called, it overwrites the existing config (input) file. So these methods should be called logically given how the pipeline is run.

The following example will run the Simulation and Training stages first with their input/output properly connected, then run the LCfitting, Getmu, and Cosmofit stages. Since to glue Training and Lcfitting (lcfitting using the trained model), the training code need to be run first so that the trained model files exist.

```
def test_pipeline():
    pipe = SALT3pipe(finput='sampleinput.txt')
    pipe.build(data=False,mode='customize',onlyrun=['byosed','sim','train'])
    pipe.configure()
    pipe.glue(['sim','train'])
    pipe.run()
    pipe.build(data=False,mode='customize',onlyrun=['lcfit','getmu','cosmofit'])
    pipe.configure()
    pipe.glue(['train','lcfit'],on='model')
    pipe.glue(['sim','lcfit'],on='phot')
    pipe.glue(['lcfit','getmu'])
    pipe.glue(['getmu','cosmofit'])
    pipe.run()
```

4.6.4 Running the Pipeline using the *runpipe.py* utility [batch submission supported]

Currently the *runpipe.py* utility is under *salt3/pipeline/*. We plan to pre-install it in the future.

Using *runpipe.py*

To use the utility, first define the environmental variable *MY_SALT3_DIR*:

```
export MY_SALT3_DIR='THE_SALT3_DIRECTORY'
```

Then in the terminal call:

```
python $MY_SALT3_DIR/SALT3/salt3/pipeline/runpipe.py -[OPTIONS] [OPTVALUES]
```

To see the currently available options, use

```
python $MY_SALT3_DIR/SALT3/salt3/pipeline/runpipe.py --help
```

```
usage: runpipe.py [-h] [-c PIPEINPUT] [--mypipe MYPIPE]
                  [--batch_mode BATCH_MODE] [--batch_script BATCH_SCRIPT]
                  [--randseed RANDSEED] [--fseeds FSEEDS] [--num NUM]
                  [--norun]

Run SALT3 Pipe.

optional arguments:
  -h, --help            show this help message and exit
  -c PIPEINPUT          pipeline input file
  --mypipe MYPIPE       define your own pipe in yourownfilename.py
  --batch_mode BATCH_MODE
                        >0 to specify how many batch jobs to submit
  --batch_script BATCH_SCRIPT
                        base batch submission script
  --randseed RANDSEED   [internal use] specify randseed for single simulation
  --fseeds FSEEDS       provide a list of randseeds for multiple batch jobs
  --num NUM             [internal use] suffix for multiple batch jobs
  --norun               set to only check configurations without launch jobs
```

Define your own pipeline

Define your own pipeline is supported by *runpipe.py*.

Simply write your own pipeline in a *MYPIPE.py* (name can be arbitrary) file and use the *–mypipe MYPIPE* flag when calling the program. Make sure to drop the *pipe.run()* line, the pipeline will be called and run in the program. Example *MYPIPE.py* file:

```
def MyPipe(finput,**kwargs):
    from pipeline import SALT3pipe
    # write your own pipeline here
    pipe = SALT3pipe(finput)
    pipe.build(data=False,mode='customize',onlyrun=['byosed','sim','train','lcfit'])
    pipe.configure()
    pipe.glue(['sim','train'])
    pipe.glue(['sim','lcfit'])
    return pipe
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search