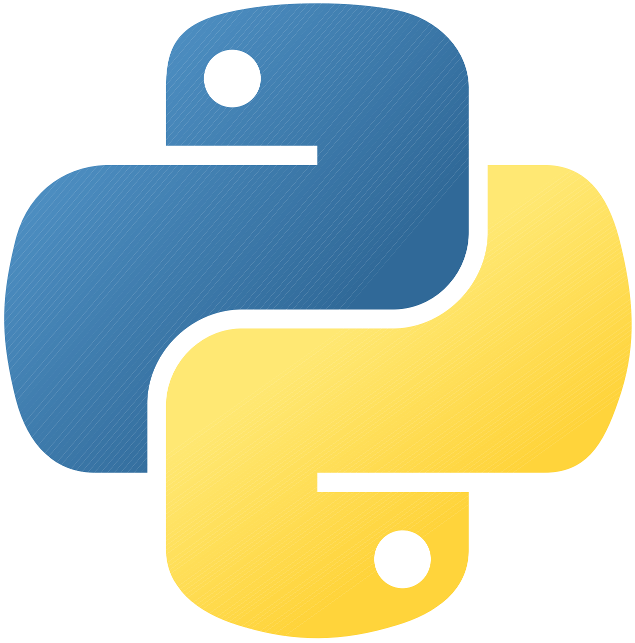
sncosmo

Python library for supernova cosmology

Kyle Barbary

UC Berkeley / BIDS · github.com/kbarbary · @kylebarbary

Yes, this talk is about



Python

But more broadly:

But more broadly:

1. Generic components can support multiple SN models

But more broadly:

1. Generic components can support multiple SN models

   Implement a new model, reuse other architecture

But more broadly:

1. Generic components can support multiple SN models

   Implement a new model, reuse other architecture

2. Libraries > Programs

But more broadly:

1. Generic components can support multiple SN models

   Implement a new model, reuse other architecture

2. Libraries > Programs

   Expose and document an API: functions and classes

What is SNCosmo?

“Python library for supernova cosmology”

What is SNCosmo?

“Python library for ~~supernova cosmology~~”

empirical models of
supernova
spectral timeseries

What is SNCosmo?

“Python library for ~~supernova cosmology~~”

empirical models of
~~supernova~~ transient
spectral timeseries

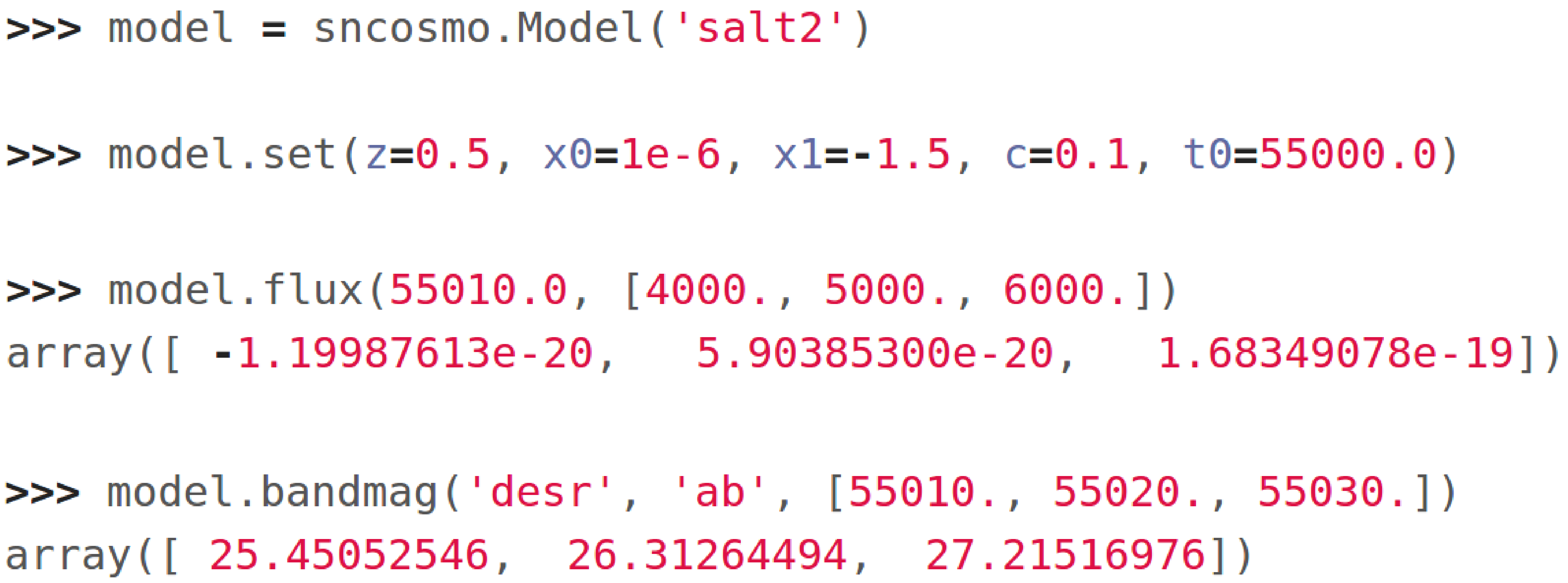
Example:

```
>>> model = sncosmo.Model('salt2')

>>> model.set(z=0.5, x0=1e-6, x1=-1.5, c=0.1, t0=55000.0)
```

Example:

```
>>> model = sncosmo.Model('salt2')

>>> model.set(z=0.5, x0=1e-6, x1=-1.5, c=0.1, t0=55000.0)

>>> model.flux(55010.0, [4000., 5000., 6000.])
array([ -1.19987613e-20,   5.90385300e-20,   1.68349078e-19])
```

Example:

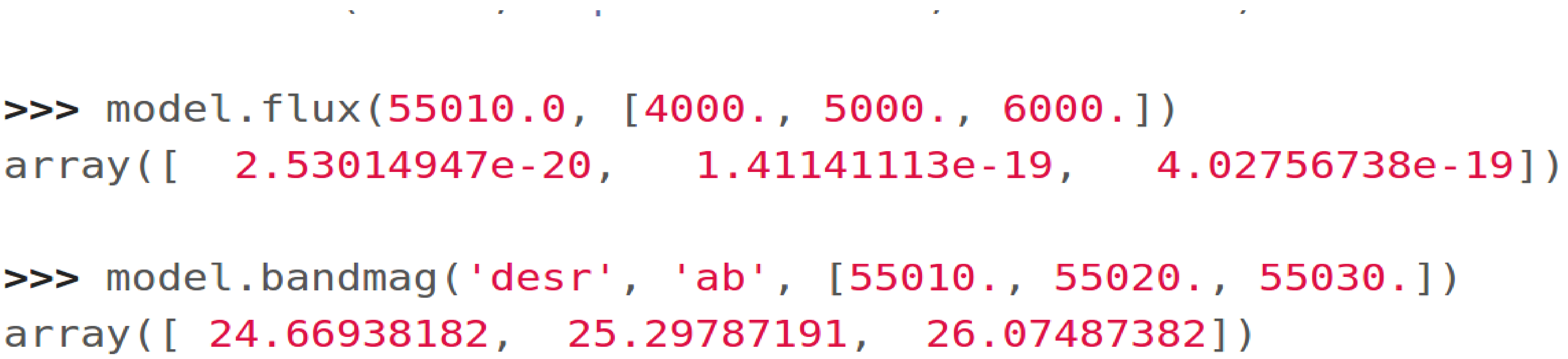


```
>>> model = sncosmo.Model('salt2')

>>> model.set(z=0.5, x0=1e-6, x1=-1.5, c=0.1, t0=55000.0)

>>> model.flux(55010.0, [4000., 5000., 6000.])
array([ -1.19987613e-20,   5.90385300e-20,   1.68349078e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 25.45052546,  26.31264494,  27.21516976])
```

Different SN models:

```
>>> model = sncosmo.Model('hsiao')

>>> model.set(z=0.5, amplitude=1e-10, t0=55000.0)

>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

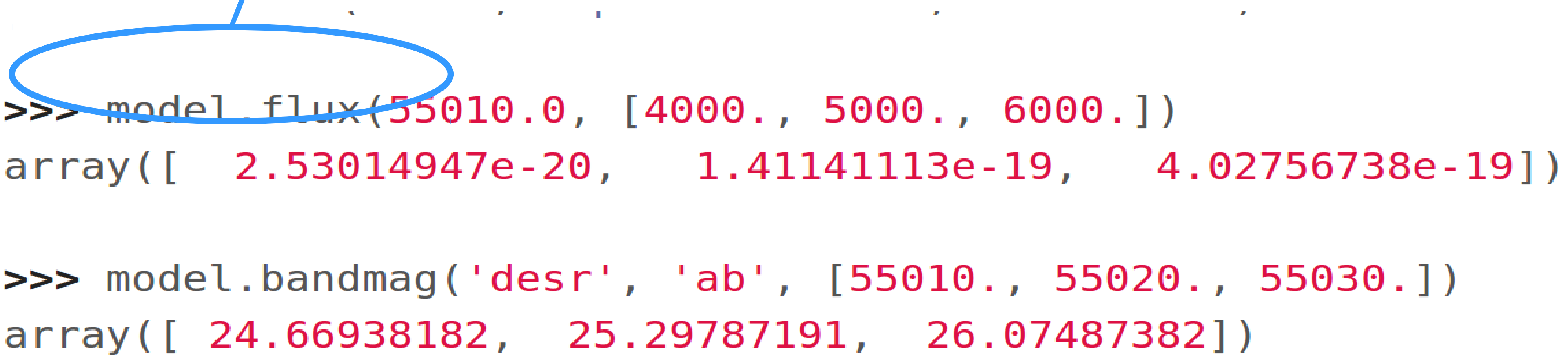
Basic components:



```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Basic components:

Model implementation

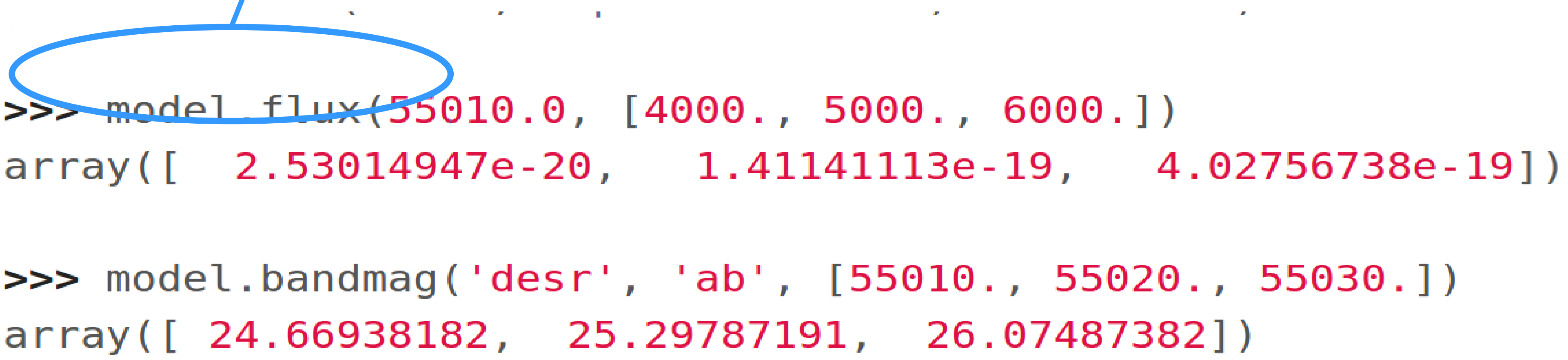


```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Basic components:

Model implementation

- SALT2, stretch, …

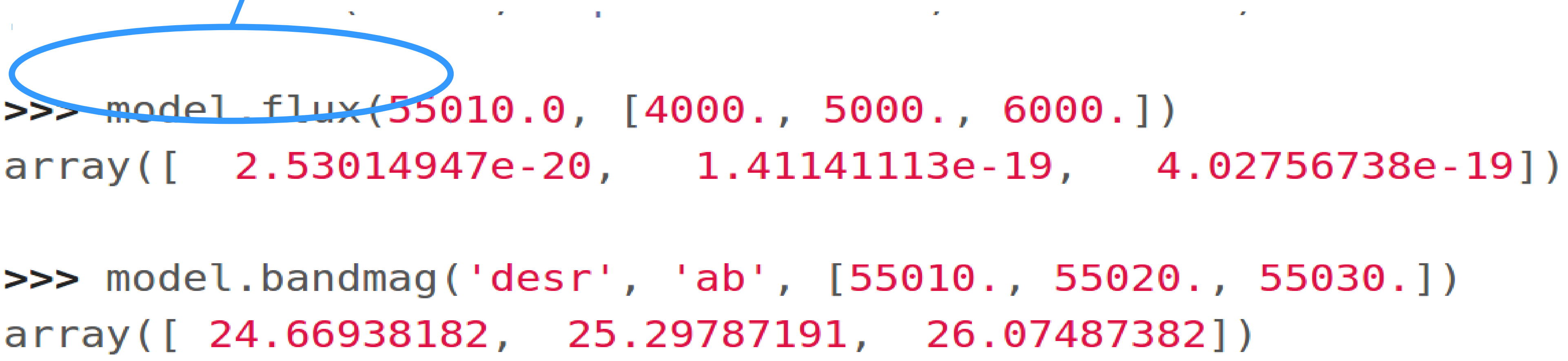


```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Basic components:

Model implementation

- SALT2, stretch, …
- redshifting and time dilation

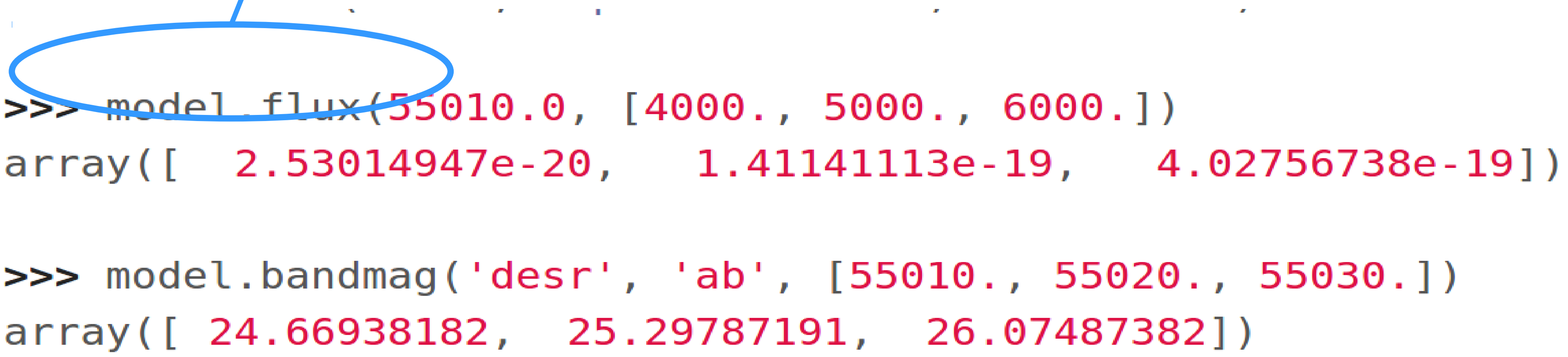


```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Basic components:

Model implementation

- SALT2, stretch, …
- redshifting and time dilation
- extinction laws

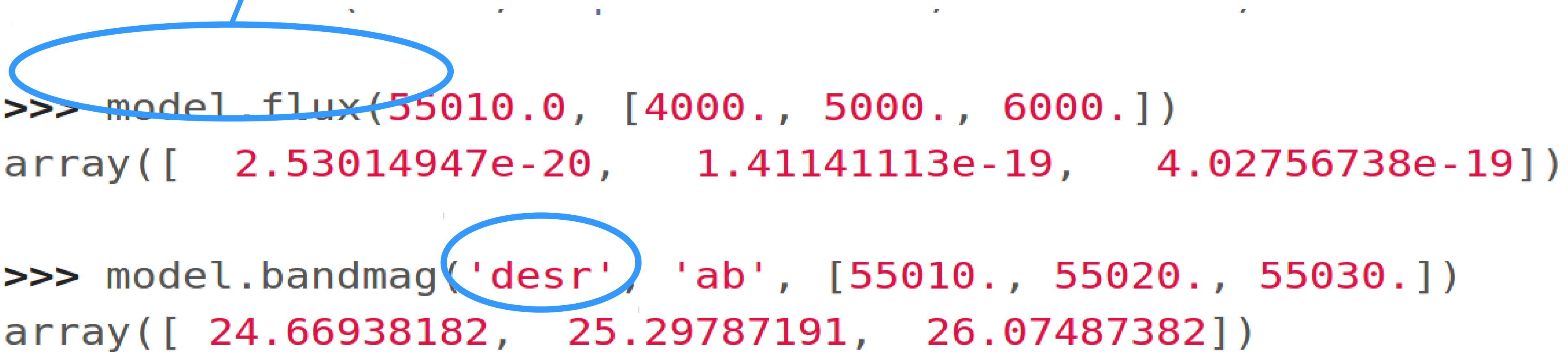


```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Basic components:

Model implementation

- SALT2, stretch, …
- redshifting and time dilation
- extinction laws



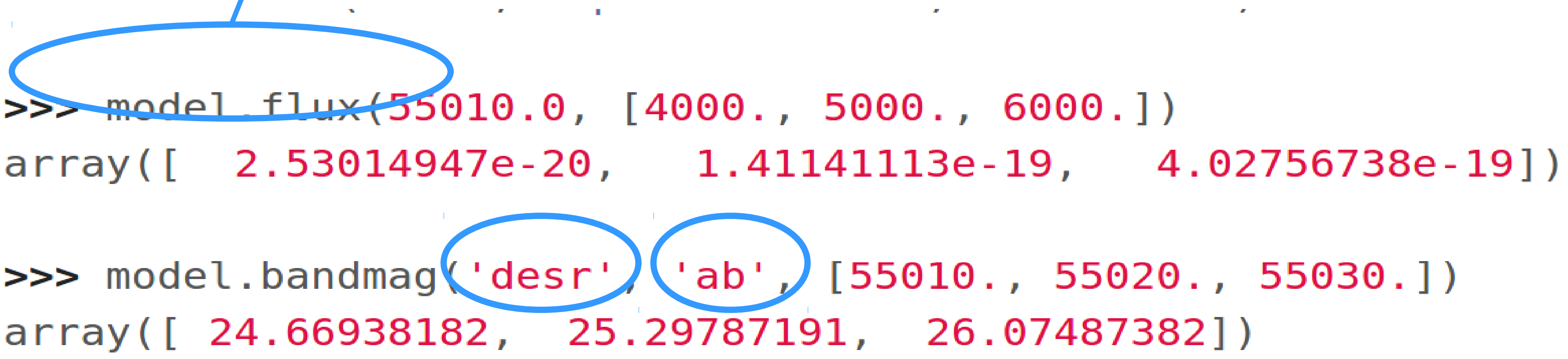
```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Bandpasses

Basic components:

Model implementation

- SALT2, stretch, …
- redshifting and time dilation
- extinction laws



```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

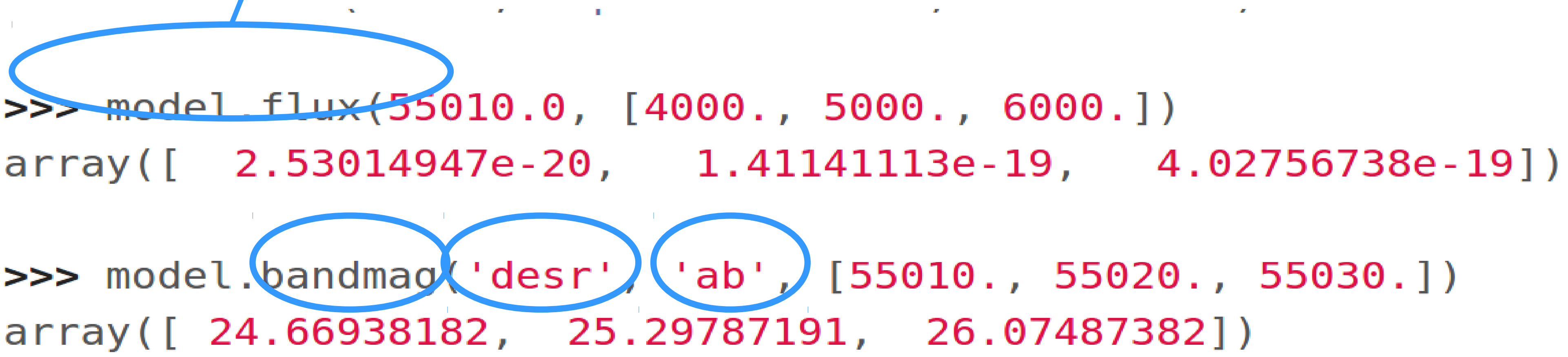
Bandpasses

Magnitude systems

Basic components:

Model implementation

- SALT2, stretch, …
- redshifting and time dilation
- extinction laws



```
>>> model.flux(55010.0, [4000., 5000., 6000.])
array([  2.53014947e-20,   1.41141113e-19,   4.02756738e-19])

>>> model.bandmag('desr', 'ab', [55010., 55020., 55030.])
array([ 24.66938182,  25.29787191,  26.07487382])
```

Integration Bandpasses Magnitude systems

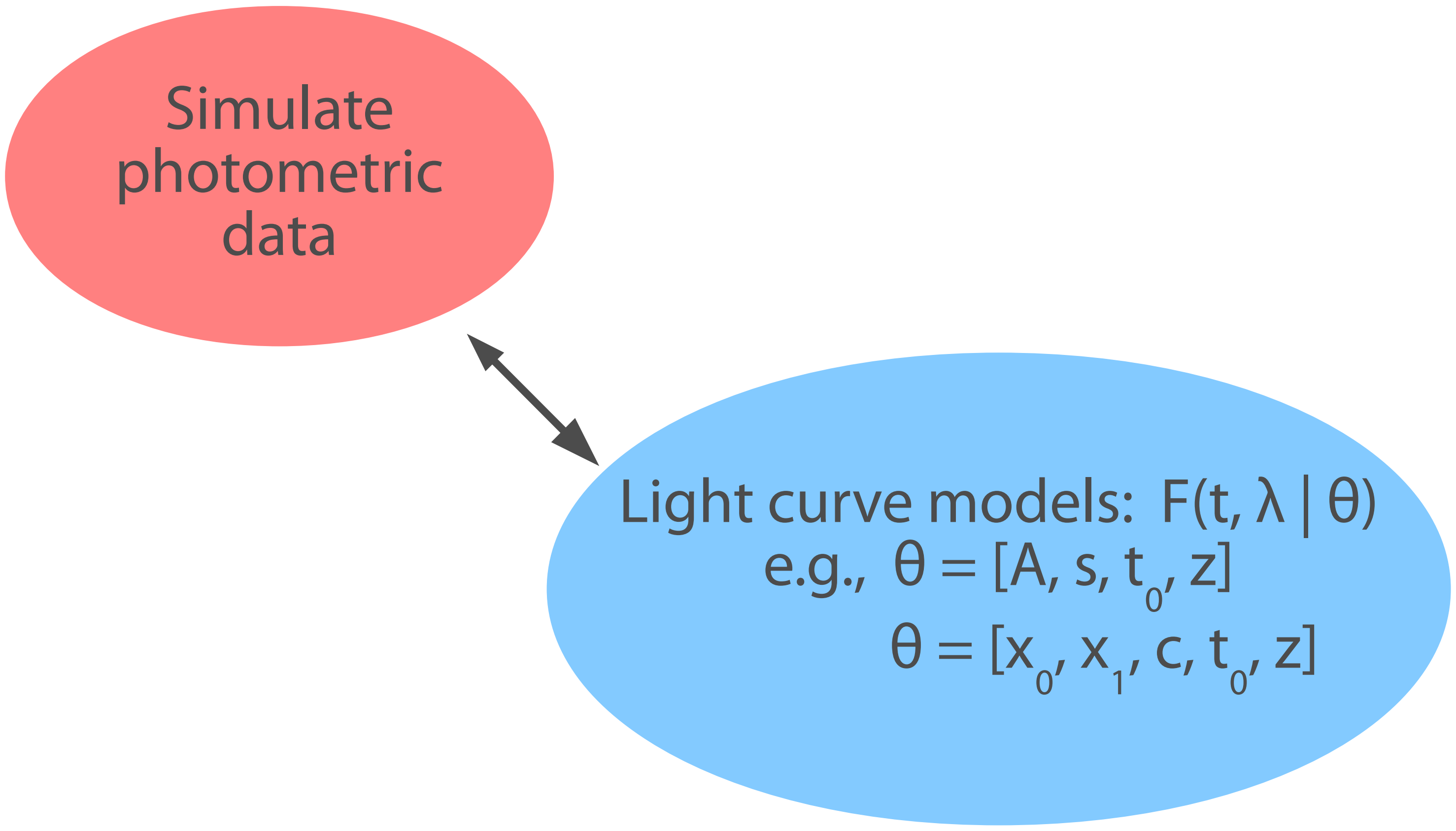
Fitting, plotting, simulation...

Light curve models: $F(t, \lambda \mid \theta)$

e.g., $\theta = [A, s, t_0, z]$

$\theta = [x_0, x_1, c, t_0, z]$

Fitting, plotting, simulation...



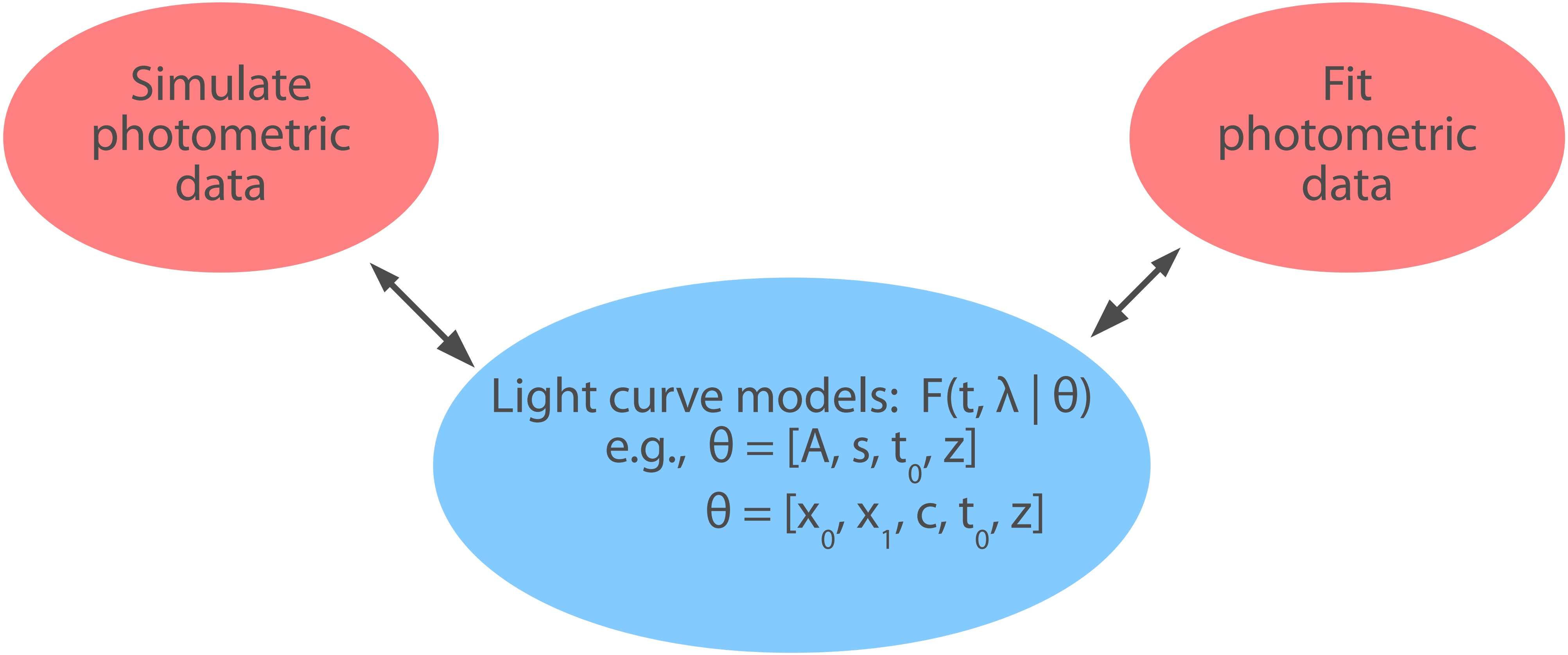
Simulate photometric data

Light curve models: $F(t, \lambda \mid \theta)$

e.g., $\theta = [A, s, t_0, z]$

$\theta = [x_0, x_1, c, t_0, z]$

Fitting, plotting, simulation...



Simulate photometric data

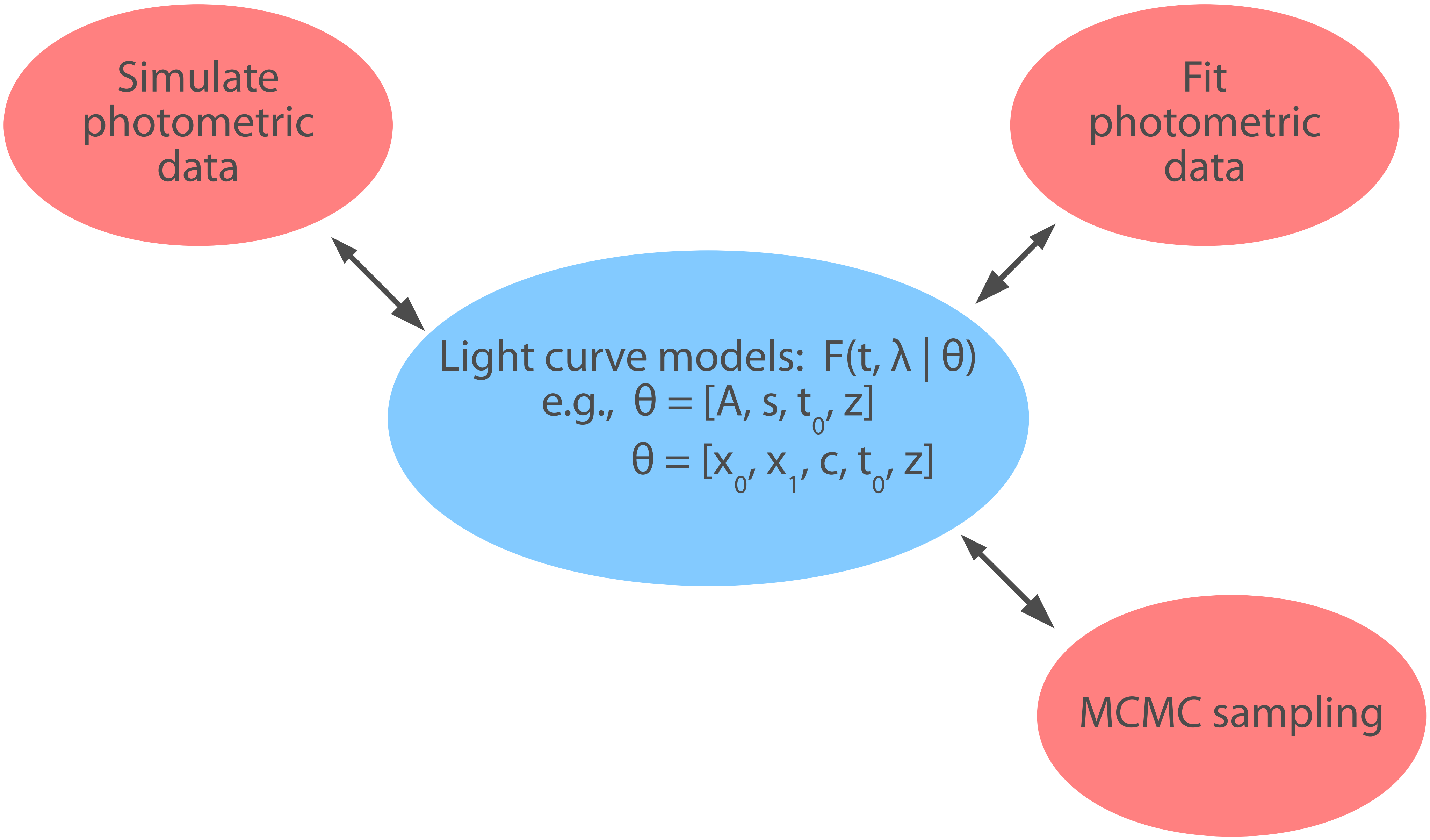
Fit photometric data

Light curve models: $F(t, \lambda \mid \theta)$

e.g., $\theta = [A, s, t_0, z]$

$\theta = [x_0, x_1, c, t_0, z]$

Fitting, plotting, simulation...



Simulate photometric data

Fit photometric data

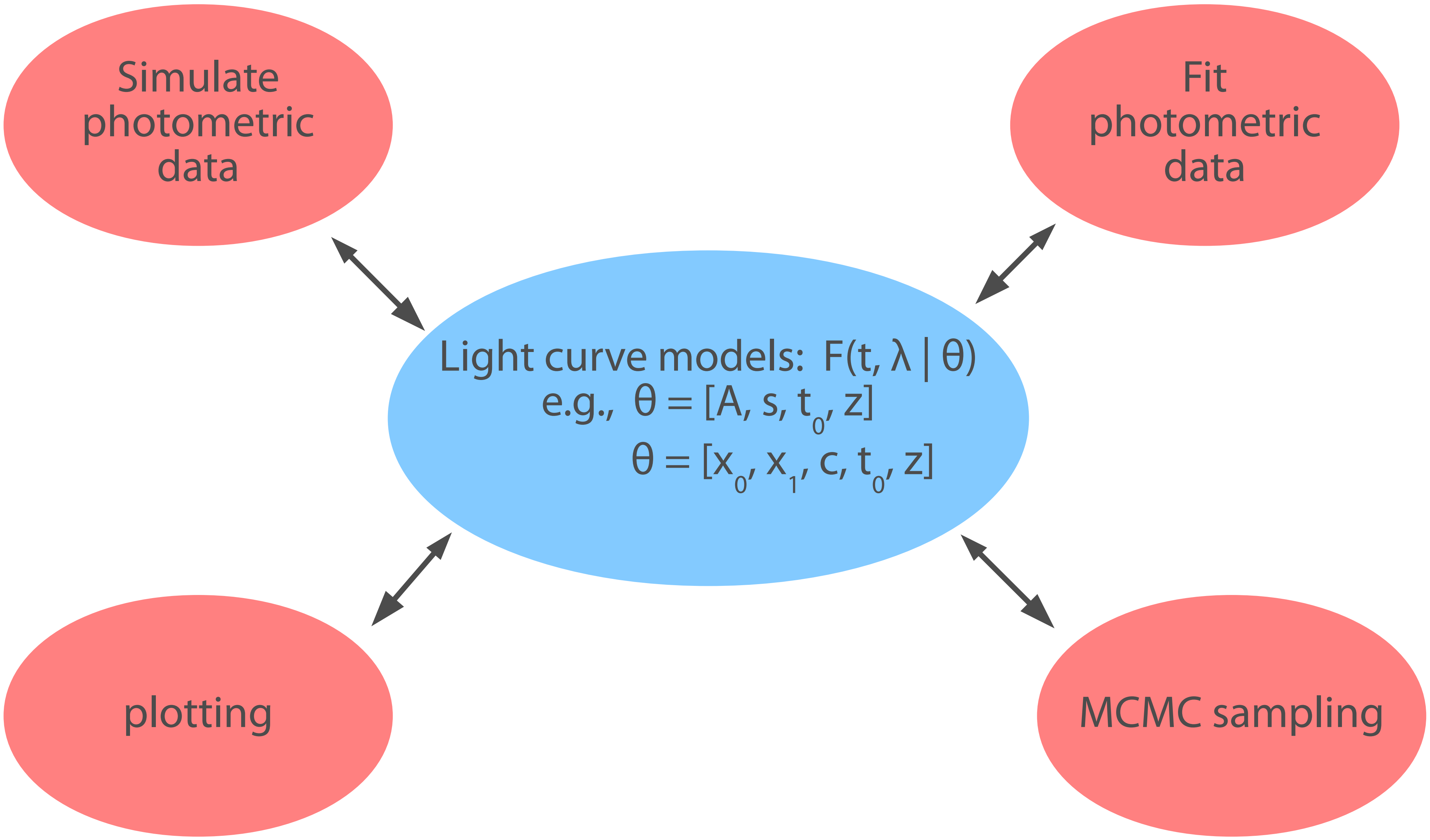
Light curve models: F(t, λ | θ)

e.g., $\theta = [A, s, t_0, z]$

$\theta = [x_0, x_1, c, t_0, z]$

MCMC sampling

Fitting, plotting, simulation...



Simulate photometric data

Fit photometric data

Light curve models: $F(t, \lambda \mid \theta)$

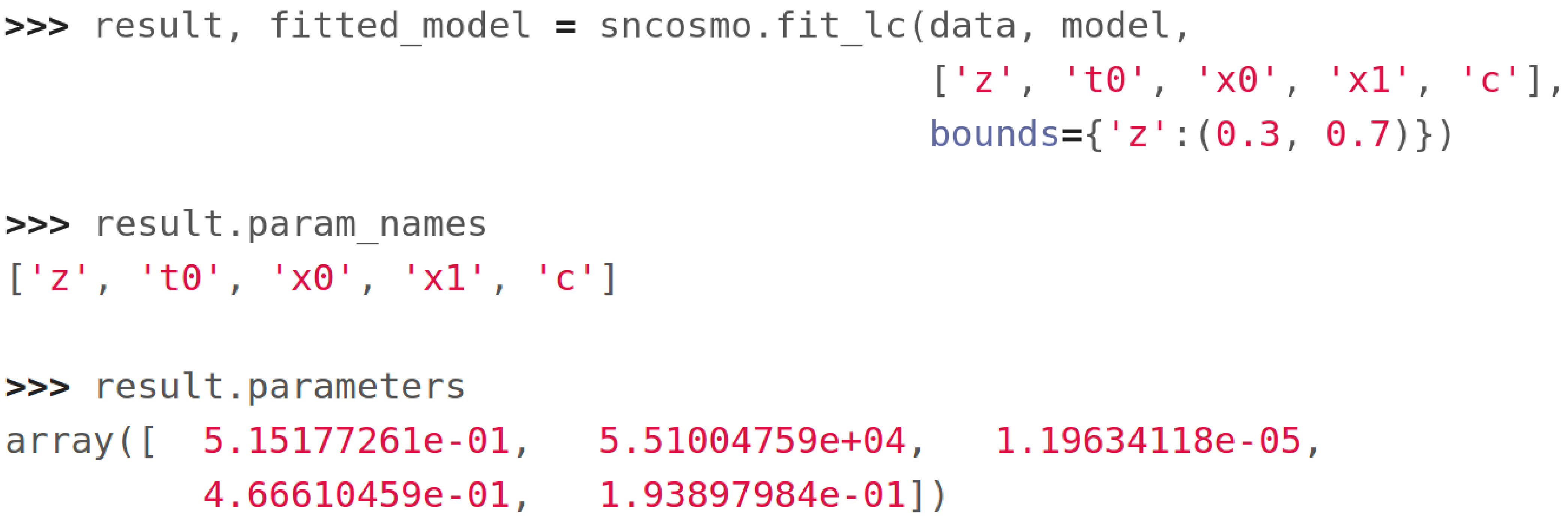
e.g., $\theta = [A, s, t_0, z]$

$\theta = [x_0, x_1, c, t_0, z]$

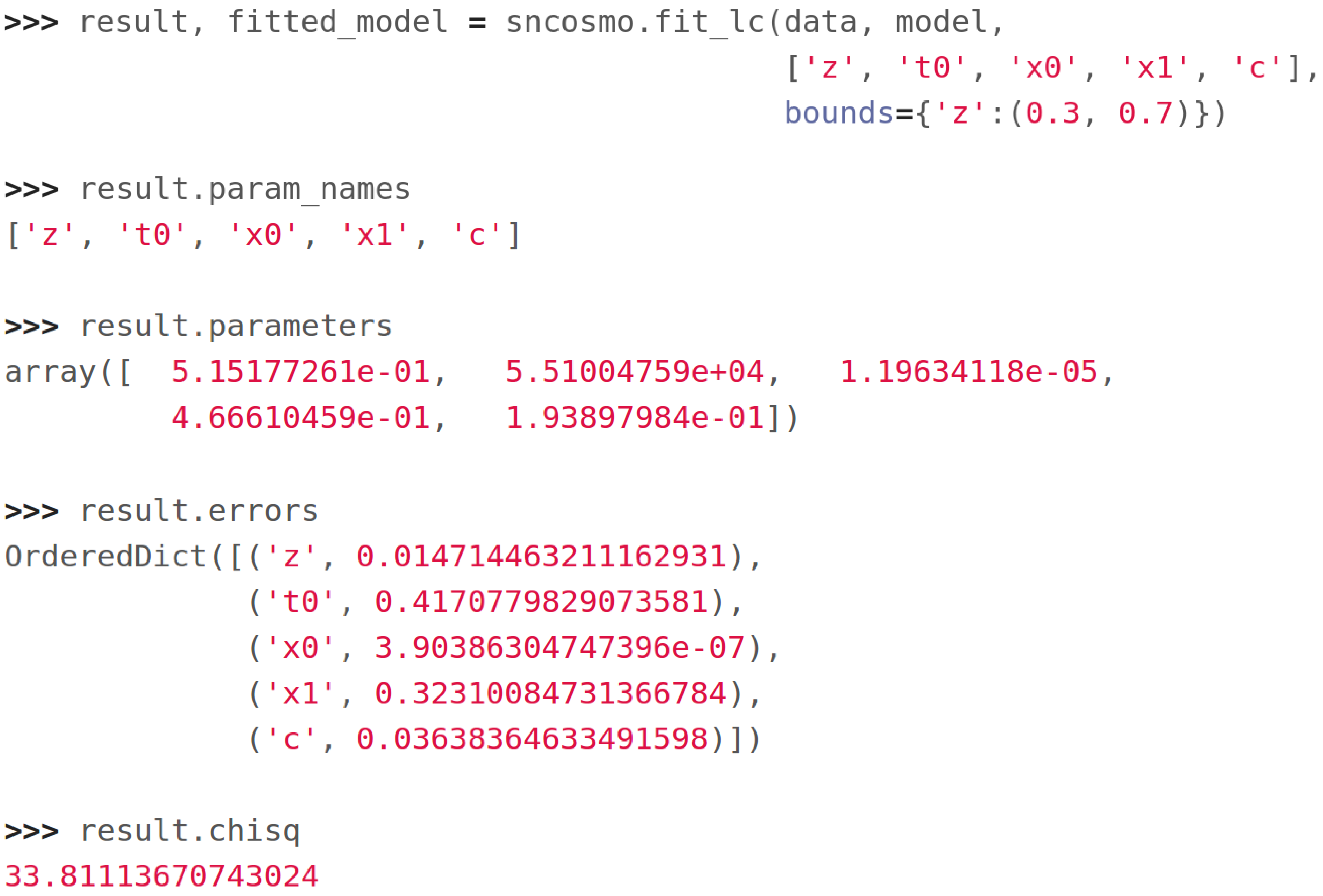
plotting

MCMC sampling

```python
>>> result, fitted_model = sncosmo.fit_lc(data, model,
                                          ['z', 't0', 'x0', 'x1', 'c'],
                                          bounds={'z':(0.3, 0.7)})
```

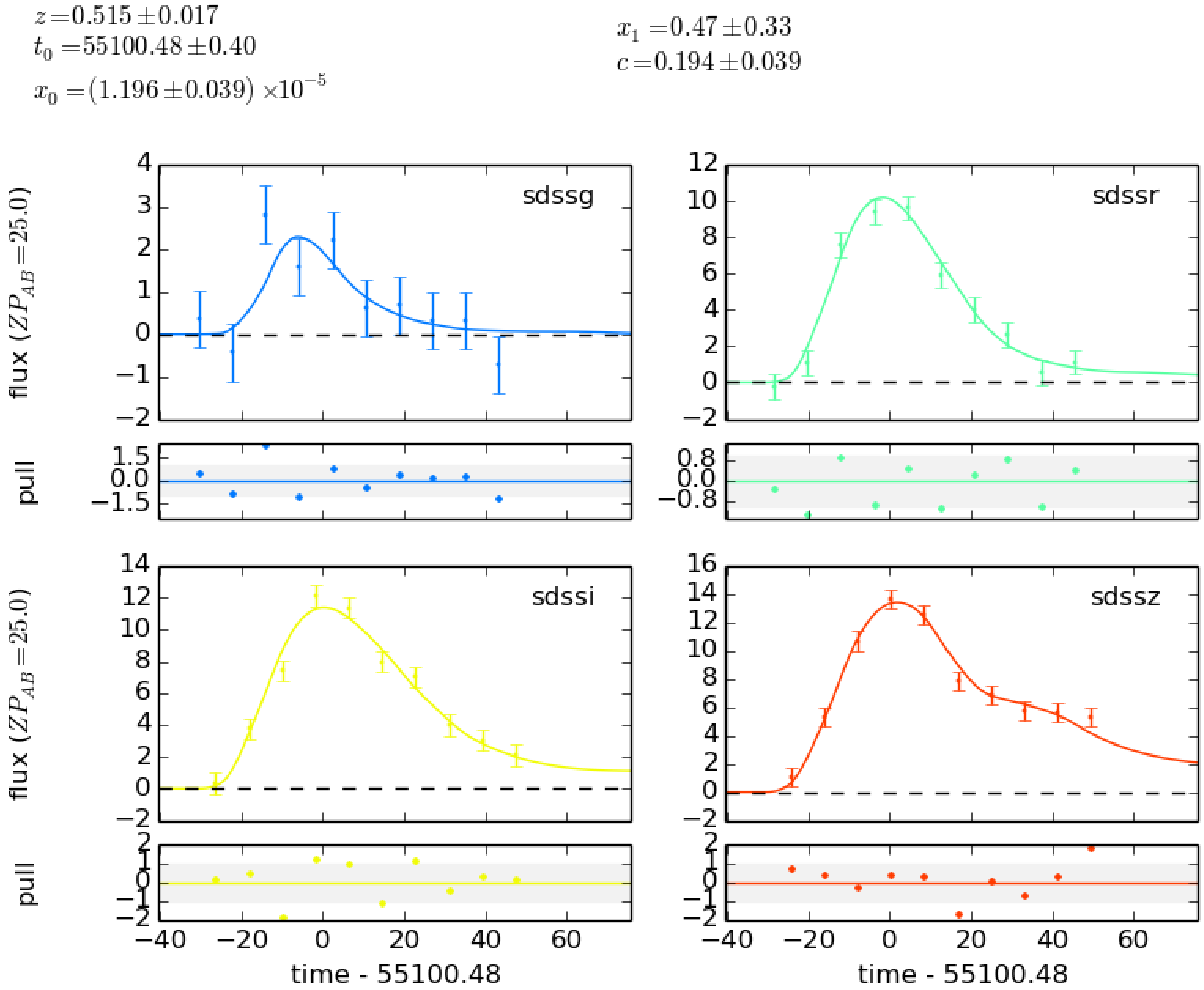


```
>>> result, fitted_model = sncosmo.fit_lc(data, model,
                                          ['z', 't0', 'x0', 'x1', 'c'],
                                          bounds={'z':(0.3, 0.7)})

>>> result.param_names
['z', 't0', 'x0', 'x1', 'c']

>>> result.parameters
array([  5.15177261e-01,   5.51004759e+04,   1.19634118e-05,
         4.66610459e-01,   1.93897984e-01])
```

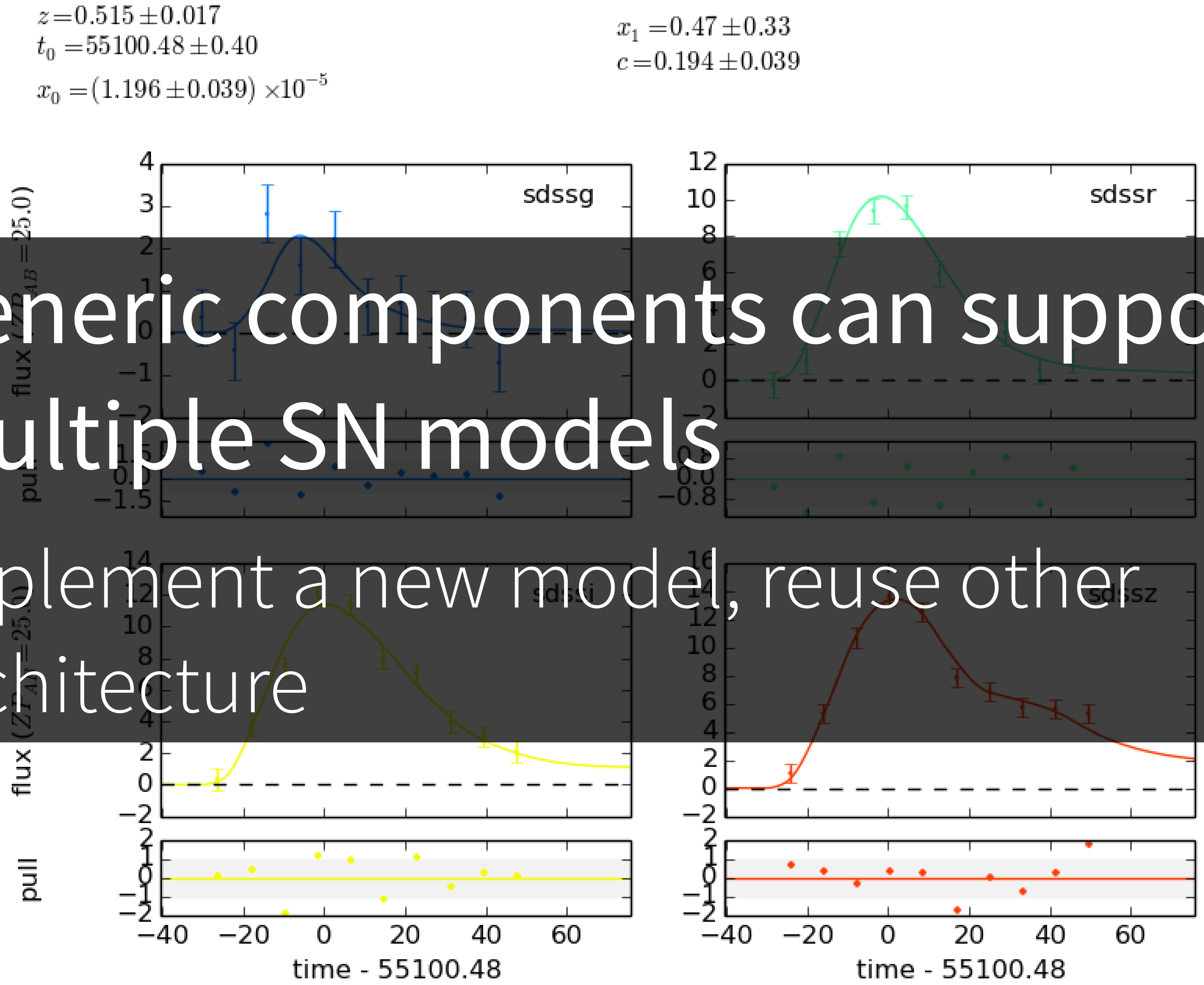


```
>>> result, fitted_model = sncosmo.fit_lc(data, model,
                                          ['z', 't0', 'x0', 'x1', 'c'],
                                          bounds={'z':(0.3, 0.7)})

>>> result.param_names
['z', 't0', 'x0', 'x1', 'c']

>>> result.parameters
array([  5.15177261e-01,   5.51004759e+04,   1.19634118e-05,
         4.66610459e-01,   1.93897984e-01])

>>> result.errors
OrderedDict([('z', 0.014714463211162931),
             ('t0', 0.41707798290735810),
             ('x0', 3.9038630474739e-07),
             ('x1', 0.32310084731366784),
             ('c', 0.03638364633491598)])

>>> result.chisq
33.81113670743024
```

```
>>> sncosmo.plot_lc(data, model=fitted_model, errors=result.errors)
>>> sncosmo.plot_lc(data, model=fitted_model, errors=result.errors)
```



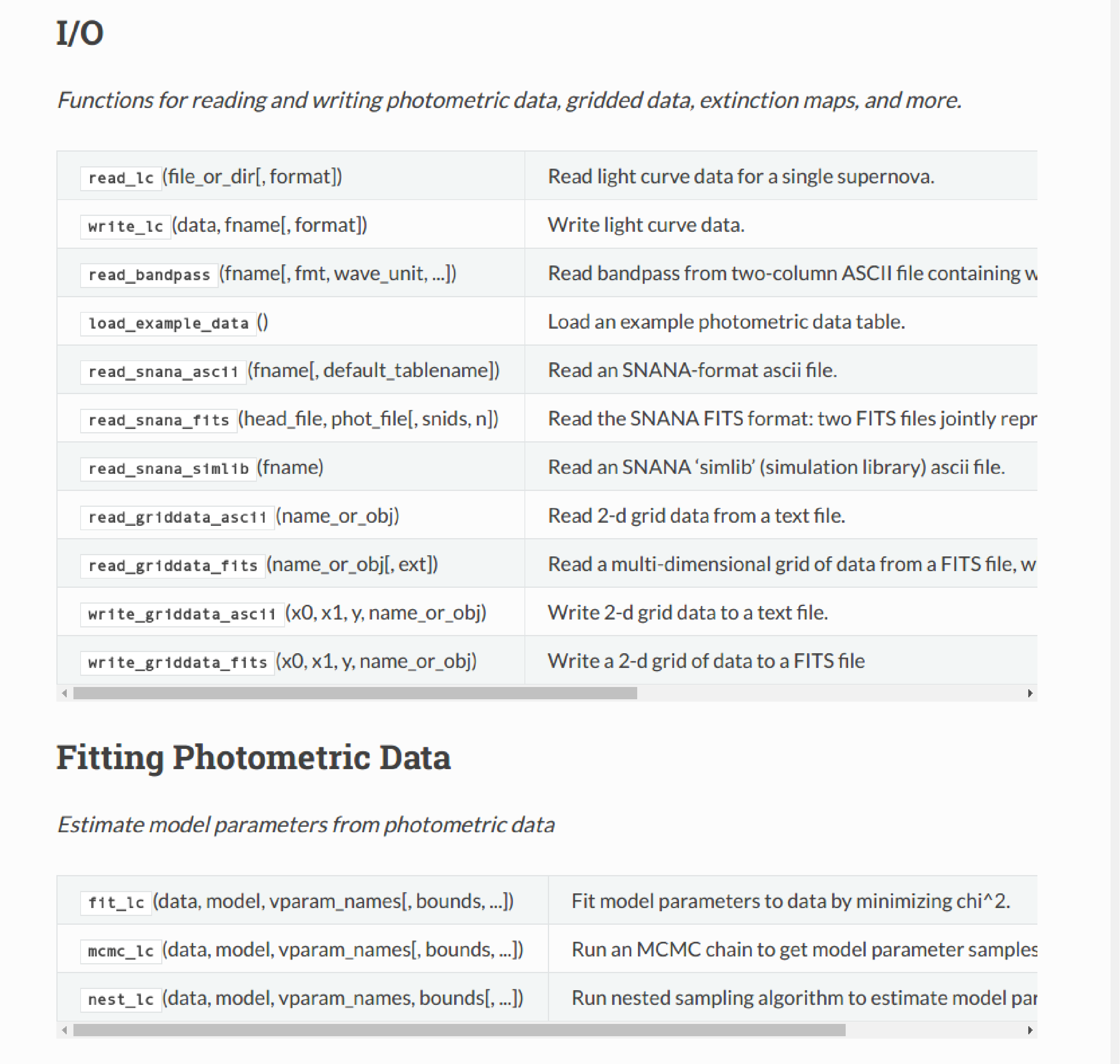
```
>>> sncosmo.plot_lc(data, model=fitted_model, errors=result.errors)
```



Generic components can support multiple SN models

Implement a new model, reuse other architecture

Documented API



I/O

Functions for reading and writing photometric data, gridded data, extinction maps, and more.

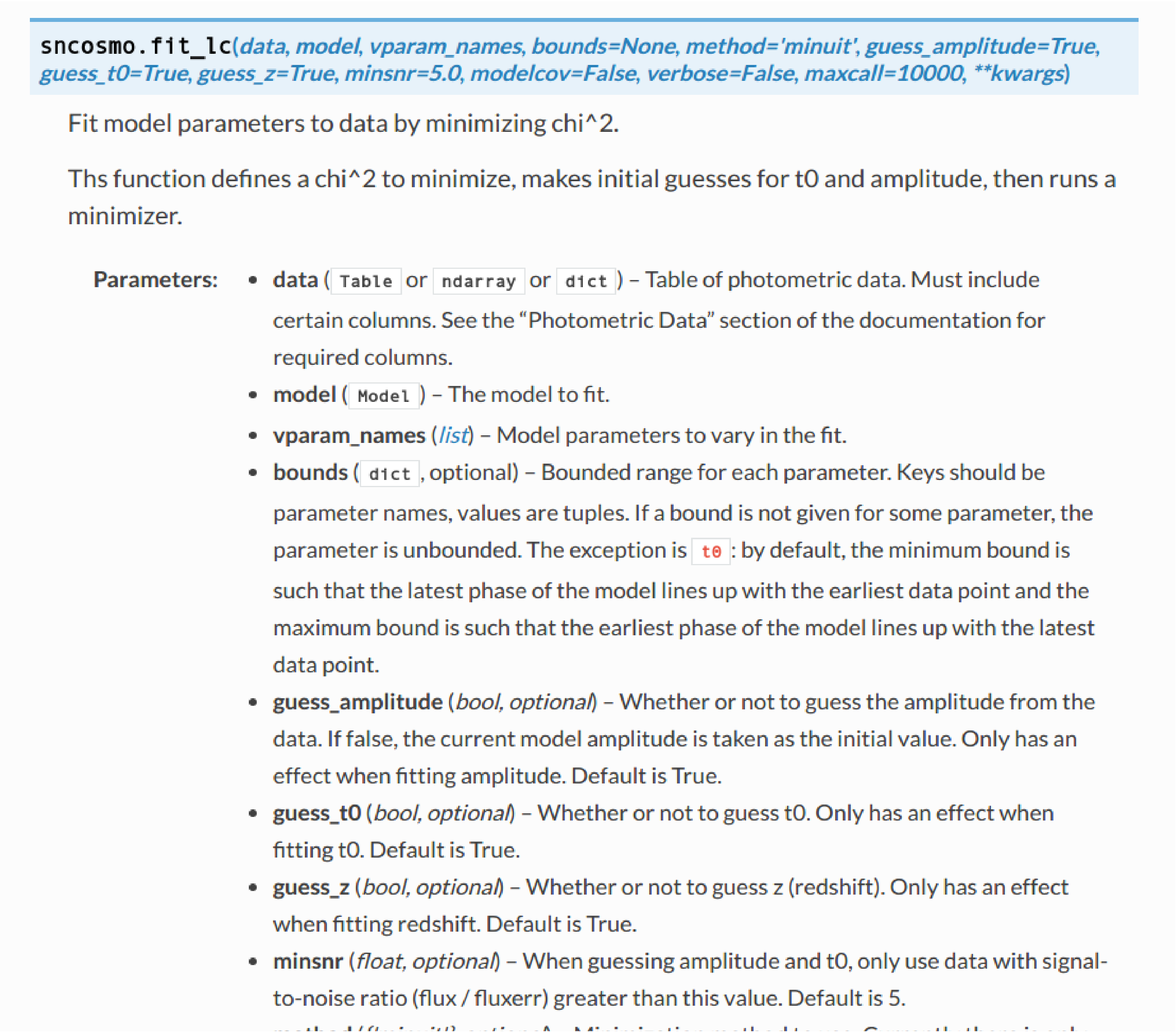
`read_lc` (file_or_dir[, format])	Read light curve data for a single supernova.
`write_lc` (data, fname[, format])	Write light curve data.
`read_bandpass` (fname[, fmt, wave_unit, ...])	Read bandpass from two-column ASCII file containing w
`load_example_data` ()	Load an example photometric data table.
`read_snana_ascii` (fname[, default_tablename])	Read an SNANA-format ascii file.
`read_snana_fits` (head_file, phot_file[, snids, n])	Read the SNANA FITS format: two FITS files jointly repr
`read_snana_simlib` (fname)	Read an SNANA 'simlib' (simulation library) ascii file.
`read_griddata_ascii` (name_or_obj)	Read 2-d grid data from a text file.
`read_griddata_fits` (name_or_obj[, ext])	Read a multi-dimensional grid of data from a FITS file, w
`write_griddata_ascii` (x0, x1, y, name_or_obj)	Write 2-d grid data to a text file.
`write_griddata_fits` (x0, x1, y, name_or_obj)	Write a 2-d grid of data to a FITS file

Fitting Photometric Data

Estimate model parameters from photometric data

`fit_lc` (data, model, vparam_names[, bounds, ...])	Fit model parameters to data by minimizing chi^2.
`mcmc_lc` (data, model, vparam_names[, bounds, ...])	Run an MCMC chain to get model parameter samples
`nest_lc` (data, model, vparam_names, bounds[, ...])	Run nested sampling algorithm to estimate model par

Documented API



`sncosmo.fit_lc`(*data, model, vparam_names, bounds=None, method='minuit', guess_amplitude=True, guess_t0=True, guess_z=True, minsnr=5.0, modelcov=False, verbose=False, maxcall=10000, **kwargs*)

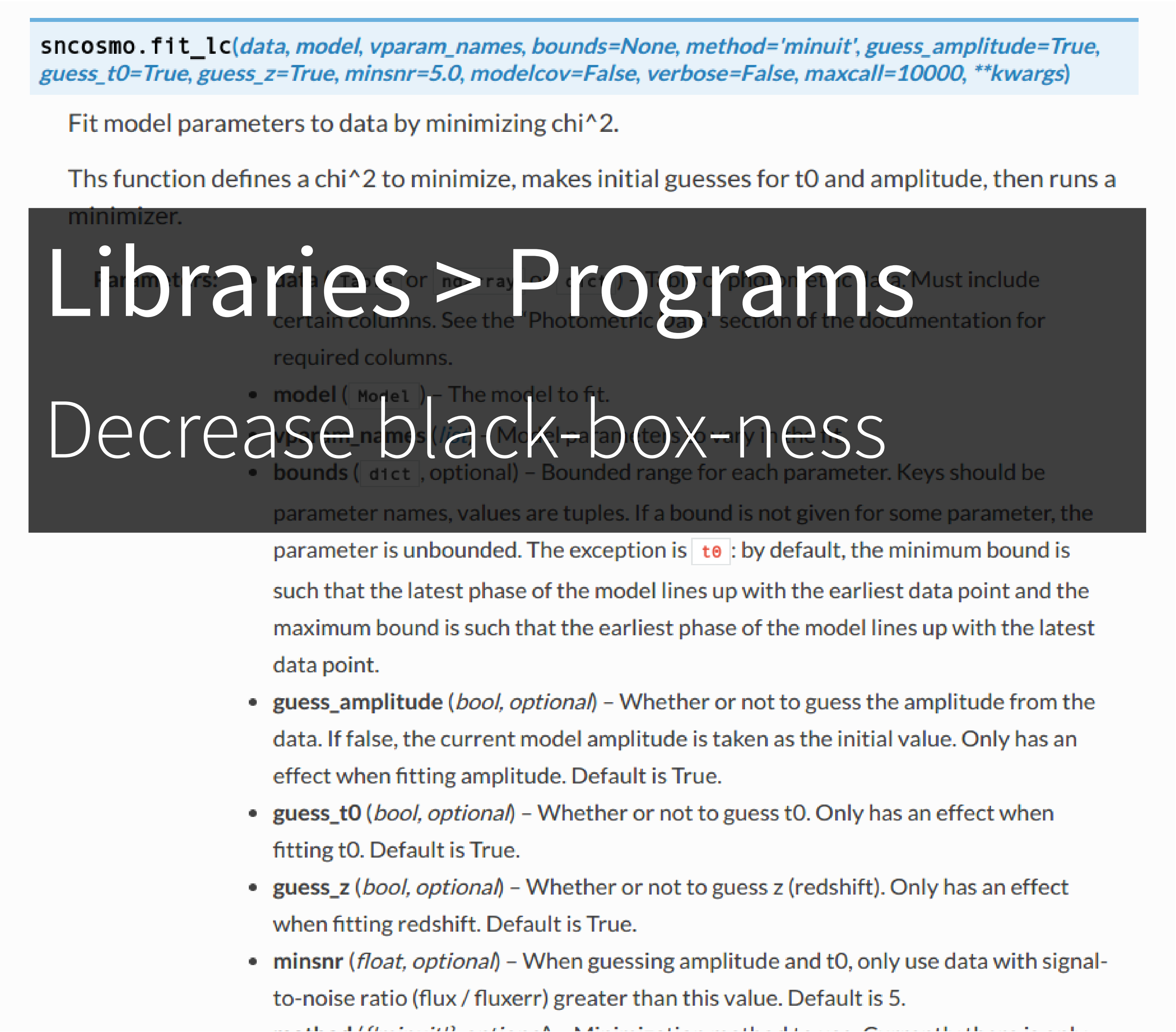
Fit model parameters to data by minimizing chi^2.

Ths function defines a chi^2 to minimize, makes initial guesses for t0 and amplitude, then runs a minimizer.

Parameters:

- **data** (`Table` or `ndarray` or `dict`) – Table of photometric data. Must include certain columns. See the “Photometric Data” section of the documentation for required columns.
- **model** (`Model`) – The model to fit.
- **vparam_names** (*list*) – Model parameters to vary in the fit.
- **bounds** (`dict`, optional) – Bounded range for each parameter. Keys should be parameter names, values are tuples. If a bound is not given for some parameter, the parameter is unbounded. The exception is `t0`: by default, the minimum bound is such that the latest phase of the model lines up with the earliest data point and the maximum bound is such that the earliest phase of the model lines up with the latest data point.
- **guess_amplitude** (*bool, optional*) – Whether or not to guess the amplitude from the data. If false, the current model amplitude is taken as the initial value. Only has an effect when fitting amplitude. Default is True.
- **guess_t0** (*bool, optional*) – Whether or not to guess t0. Only has an effect when fitting t0. Default is True.
- **guess_z** (*bool, optional*) – Whether or not to guess z (redshift). Only has an effect when fitting redshift. Default is True.
- **minsnr** (*float, optional*) – When guessing amplitude and t0, only use data with signal-to-noise ratio (flux / fluxerr) greater than this value. Default is 5.
- **method** (*{'minuit'}, optional*) – Minimization method to use. Currently there is only

Documented API



`sncosmo.fit_lc`(*data, model, vparam_names, bounds=None, method='minuit', guess_amplitude=True, guess_t0=True, guess_z=True, minsnr=5.0, modelcov=False, verbose=False, maxcall=10000, **kwargs*)

Fit model parameters to data by minimizing chi^2.

Ths function defines a chi^2 to minimize, makes initial guesses for t0 and amplitude, then runs a minimizer.

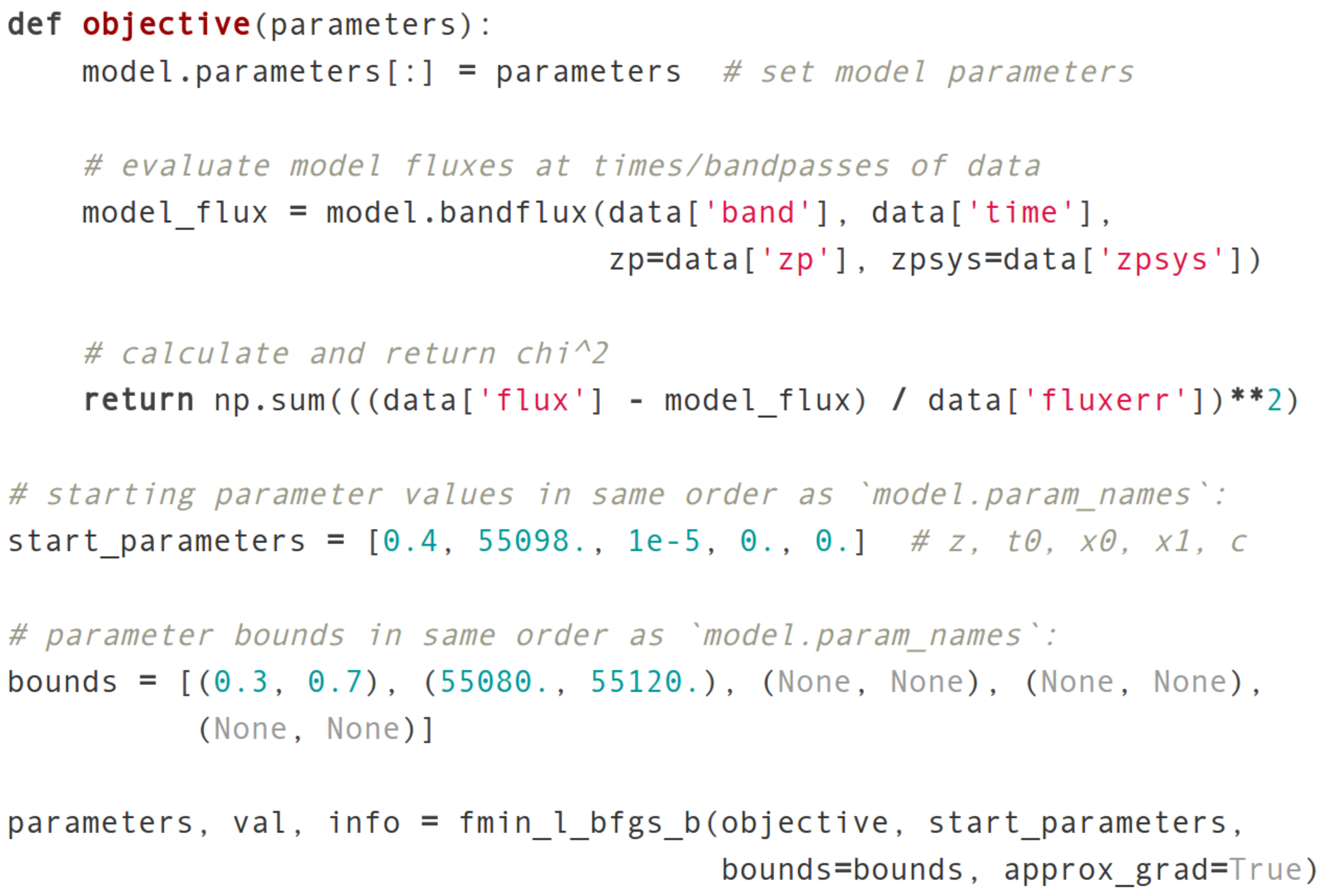
Parameters:

- **data** (`Table` or `ndarray` or `dict`) – Table of photometric data. Must include certain columns. See the 'Photometric Data' section of the documentation for required columns.
- **model** (`Model`) – The model to fit.
- **vparam_names** (*list*) – Model parameters to vary in the fit.
- **bounds** (`dict`, optional) – Bounded range for each parameter. Keys should be parameter names, values are tuples. If a bound is not given for some parameter, the parameter is unbounded. The exception is `t0`: by default, the minimum bound is such that the latest phase of the model lines up with the earliest data point and the maximum bound is such that the earliest phase of the model lines up with the latest data point.
- **guess_amplitude** (*bool, optional*) – Whether or not to guess the amplitude from the data. If false, the current model amplitude is taken as the initial value. Only has an effect when fitting amplitude. Default is True.
- **guess_t0** (*bool, optional*) – Whether or not to guess t0. Only has an effect when fitting t0. Default is True.
- **guess_z** (*bool, optional*) – Whether or not to guess z (redshift). Only has an effect when fitting redshift. Default is True.
- **minsnr** (*float, optional*) – When guessing amplitude and t0, only use data with signal-to-noise ratio (flux / fluxerr) greater than this value. Default is 5.
- **method** (*{'minuit'}, optional*) – Minimization method to use. Currently there is only

Libraries > Programs

Decrease black-box-ness

Example: custom fitter

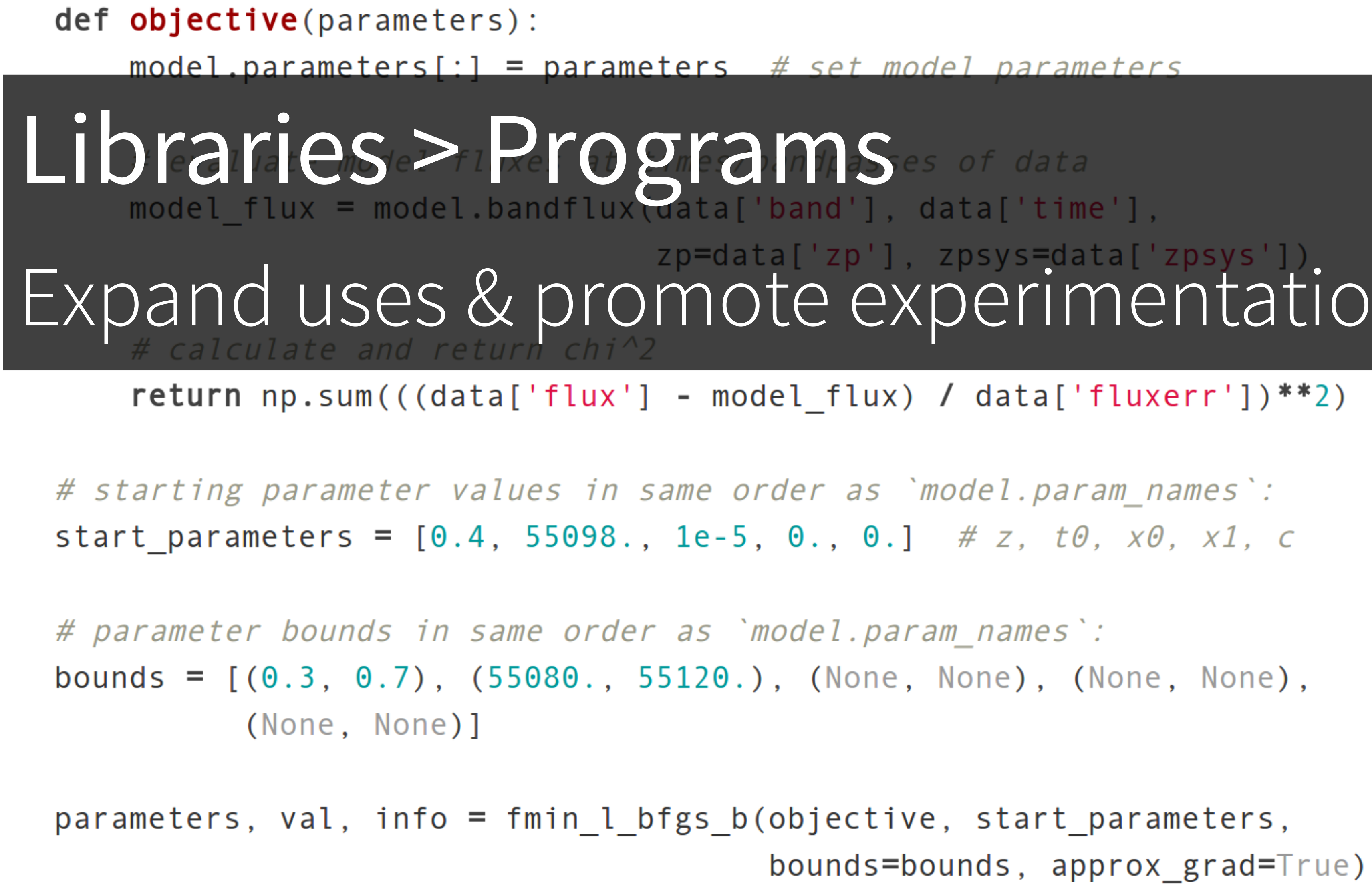


```python
def objective(parameters):
    model.parameters[:] = parameters  # set model parameters

    # evaluate model fluxes at times/bandpasses of data
    model_flux = model.bandflux(data['band'], data['time'],
                                zp=data['zp'], zpsys=data['zpsys'])

    # calculate and return chi^2
    return np.sum(((data['flux'] - model_flux) / data['fluxerr'])**2)

# starting parameter values in same order as `model.param_names`:
start_parameters = [0.4, 55098., 1e-5, 0., 0.]  # z, t0, x0, x1, c

# parameter bounds in same order as `model.param_names`:
bounds = [(0.3, 0.7), (55080., 55120.), (None, None), (None, None),
          (None, None)]

parameters, val, info = fmin_l_bfgs_b(objective, start_parameters,
                                      bounds=bounds, approx_grad=True)
```

Example: custom fitter

Libraries > Programs

Expand uses & promote experimentation



```
def objective(parameters):
    model.parameters[:] = parameters  # set model parameters

    # evaluate model fluxes at times/bandpasses of data
    model_flux = model.bandflux(data['band'], data['time'],
                                zp=data['zp'], zpsys=data['zpsys'])

    # calculate and return chi^2
    return np.sum(((data['flux'] - model_flux) / data['fluxerr'])**2)

# starting parameter values in same order as `model.param_names`:
start_parameters = [0.4, 55098., 1e-5, 0., 0.]  # z, t0, x0, x1, c

# parameter bounds in same order as `model.param_names`:
bounds = [(0.3, 0.7), (55080., 55120.), (None, None), (None, None),
          (None, None)]

parameters, val, info = fmin_l_bfgs_b(objective, start_parameters,
                                      bounds=bounds, approx_grad=True)
```

Example: custom fitter

```
def objective(parameters):
    model.parameters[:] = parameters  # set model parameters

    # evaluate model fluxes at times/bandpasses of data
    model_flux = model.bandflux(data['band'], data['time'],
                                zp=data['zp'], zpsys=data['zpsys'])

    # calculate and return chi^2
    return np.sum(((data['flux'] - model_flux) / data['fluxerr'])**2)

# starting parameter values in same order as `model.param_names`:
start_parameters = [0.4, 55098., 1e-5, 0., 0.]  # z, t0, x0, x1, c

# parameter bounds in same order as `model.param_names`:
bounds = [(0.3, 0.7), (55080., 55120.), (None, None), (None, None),
          (None, None)]

parameters, val, info = fmin_l_bfgs_b(objective, start_parameters,
                                      bounds=bounds, approx_grad=True)
```

Libraries > Programs

Expand uses & promote experimentation

Caveats...

Harder to maintain

Not appropriate for more experimental code

Harder to use (can build executable on top)

http://sncosmo.readthedocs.io
http://github.com/sncosmo/sncosmo

http://sncosmo.readthedocs.io

http://github.com/sncosmo/sncosmo

Thanks!