symlens Documentation

Release 0.3.3

Mathew Madhavacheril

Jul 25, 2020

Contents:

This library allows one to build and evaluate arbitrary separable mode-coupling estimators. In practice, its main purpose is to provide a flat-sky lensing estimator code. More generally, one can build estimators and noise functions for convergence, magnification, shear, mixed estimators (for gradient cleaning), split-based lensing, birefringence, patchy tau, etc. and cross-covariances between these.

Instead of having to calculate by hand the separable forms of the above, one simply provides the mode-coupling and filter expressions, and a `sympy`-based (Mathematica-like) backend factorizes these expressions into FFT-only form (i.e., no explicit convolutions are required).

Curved sky support does not exist. Adding it is possibly non-trivial, but thoughts and ideas (and PRs!) are highly appreciated. Still, this package can serve as the backend for quick exploration of various kinds of estimators.

CHAPTER 1

1 symlens

This library allows one to build and evaluate arbitrary separable mode-coupling estimators. In practice, its main purpose is to provide a flat-sky lensing estimator code. More generally, one can build estimators and noise functions for convergence, magnification, shear, mixed estimators (for gradient cleaning), split-based lensing, birefringence, patchy tau, etc. and cross-covariances between these.

Instead of having to calculate by hand the separable forms of the above, one simply provides the mode-coupling and filter expressions, and a `sympy`-based (Mathematica-like) backend factorizes these expressions into FFT-only form (i.e., no explicit convolutions are required).

Curved sky support does not exist. Adding it is possibly non-trivial, but thoughts and ideas (and PRs!) are highly appreciated. Still, this package can serve as the backend for quick exploration of various kinds of estimators.

- Free software: BSD license
- Documentation: https://symlens.readthedocs.io.

1.1 1.1 Dependencies

- Python>=3.6
- pixell
- numpy, sympy

1.2 1.2 Installing

To install, run:

```
$ python setup.py install --user
```

1.3 1.3 Usage

See the Usage guide and the API Reference.

An important thing to remember is that by default, the code expects "physical" normalization of FFTs in pixell (not the default normalization in pixell), i.e. you should be passing in Fourier maps that come from something like

```
kmap = enmap.fft(imap,normalize='phys')
```

or

```
kmaps = enmap.map2harm(imaps,normalize='phys')
```

1.3.1 1.3.1 Contributing

If you have write access to this repository, please:

1. create a new branch
2. push your changes to that branch
3. merge or rebase to get in sync with master
4. submit a pull request on github

If you do not have write access, create a fork of this repository and proceed as described above.

CHAPTER 2

2 Usage

2.1 2.1 Basic usage

The most common use case for `symlens` is to get noise curves for lensing, and to run lensing reconstruction on CMB maps. We will cover this first, and later come to custom estimators that exploit the full power and flexibility of this package.

2.1.1 2.1.1 Geometry

Skip this if you are familiar with the pixell library. Most functions that return something immediately useful take a `shape`,‘‘wcs‘‘ pair as input. In short, this pair specifies the footprint or geometry of the patch of sky on which calculations are done. If you are playing with simulations or just want quick forecasts, you can make your own `shape`,‘‘wcs‘‘ pair as follows:

```
>>> from pixell import enmap
>>> shape,wcs =
enmap.geometry(shape=(2048,2048),res=np.deg2rad(2.0/60.),pos=(0,0))
```

to specify a footprint consisting of 2048 x 2048 pixels of width 2 arcminutes and centered on the origin of a CEA geometry.

All outputs are 2D arrays in Fourier space, so you will need some way to bin it in annuli. A map of the absolute wavenumbers is useful for this

```
>>> modlmap = enmap.modlmap(shape,wcs)
```

To read a map in from disk and get its geometry, you could do

```
>>> imap = enmap.read_map(fits_file_name)
>>> shape,wcs = imap.shape, imap.wcs
```

Please read the documentation for pixell for more information.

2.1.2 2.1.2 Noise curves

Using the machinery described in `Custom estimators` below, a number of pre-defined mode-coupling noise curves have been built. We provide some examples of using these. All of these require a `shape`,``wcs`` geometry pair as described above. Next, you also need to have a Fourier space mask at hand that enforces what multipoles in the CMB map are included.

A convenience function is provided for generating simple Fourier space masks. e.g.

```
>>> from symlens import utils
>>> tellmin = 100
>>> tellmax = 3000
>>> kmask = utils.mask_kspace(shape,wcs,lmin=tellmin,lmax=tellmax)
```

Finally, you also need a `feed_dict`, a dictionary which maps names of variables (keys) to 2D arrays containing data, filters, etc. which are fed in at the very final integration step. With custom estimators described later, you get to choose the names of your variables. But the convenience of the canned functions described here comes with the cost of having to learn what variable convention is defined inside them. We will learn by example.

Lensing noise curves require CMB power spectra. The naming convention for the `feed_dict` for these is:

1. `uC_X_Y` for CMB XY spectra that go in the lensing response function, e.g. `uC_T_T` for the TT spectrum (despite the notation, this should be the lensed spectrum, not the unlensed spectrum)
2. `tC_X_Y` for total CMB XY spectra that go in the lensing filters. e.g. `tC_T_T` for the total TT spectrum that includes beam-deconvolved noise.

These have to be specified on the 2D Fourier space grid. We can build them like this:

```
>>> feed_dict = {}
>>> feed_dict['uC_T_T'] = utils.interp(ells,ctt)(modlmap)
>>> feed_dict['tC_T_T'] = utils.interp(ells,ctt)(modlmap)+(33.*np.pi/180./60.)**2./
↪utils.gauss_beam(modlmap,7.0)**2.
```

where I've used the convenience function `interp` to interpolate an `ells`,``cltt`` 1D spectrum specification isotropically on to the Fourier space grid, and created a Planck-like total beam-deconvolved spectrum using the `gauss_beam` function. That's it! Now we can get the pre-built Hu Okamoto 2001 (estimator="hu_ok") noise for the TT lensing estimator as follows,

```
>>> import symlens as s
>>> nl2d = s.N_l(shape,wcs,feed_dict,"hu_ok","TT",xmask=kmask,ymask=kmask)
```

which can be binned in annuli to obtain a lensing noise curve.

2.1.3 2.1.3 Lensing maps

To make a lensing map, we need to provide beam deconvolved Fourier maps of the CMB, which for a quadratic estimator <XY> have default variable names of X and Y,

```
>>> feed_dict['X'] = beam_deconvolved_fourier_T_map
>>> feed_dict['Y'] = beam_deconvolved_fourier_T_map
```

An important thing to remember is that by default, the code expects "physical" normalization of FFTs in pixell (not the default normalization in pixell), i.e. you should be passing in Fourier maps that come from something like

```
>>> beam_deconvolved_fourier_T_map = enmap.fft(imap,normalize='phys')
```

or

```
>>> beam_deconvolved_fourier_T_map = enmap.map2harm(imaps,normalize='phys')[0]
```

One can then obtain the unnormalized lensing map simply by doing,

```
>>> ukappa = s.unnormalized_quadratic_estimator(shape,wcs,
                        feed_dict,"hu_ok","TT",xmask=kmask,ymask=kmask)
```

and also obtain its normalization,

```
>>> norm = s.A_l(shape,wcs,feed_dict,"hu_ok","TT",xmask=kmask,ymask=kmask)
```

and combine into a normalized Fourier space CMB lensing convergence map,

```
>>> fkappa = norm * ukappa
```

2.1.4 2.1.4 General noise curves

To perform more complicated calculations like cross-covariances, noise for non-optimal estimators, mixed experiment estimators (for gradient cleaning), split-based lensing N0 curves, etc., we need to learn how to attach field names, which make the `feed_dict` expect more variables than what was described earlier.

Let's first show how we can obtain a general noise cross-covariance. We can for example obtain the same TT lensing noise curve as above but in a more round-about way by asking what the cross-covariance of the TT estimator is with the TT estimator itself,

```
>>> Nl =
N_l_cross(shape,wcs,feed_dict,alpha_estimator="hu_ok",alpha_XY="TT",
                              beta_estimator="hu_ok",beta_XY="TT",
                              xmask=kmask,ymask=kmask)
```

This works just like before. However, what if the instrument noise in the first leg of the estimator is uncorrelated with the noise in the second leg? Then, we need to differentiate between the four fields that appear above. We can do that by providing names for these fields.

```
>>> Nl = N_l_cross(shape,wcs,feed_dict,
             alpha_estimator="hu_ok",alpha_XY="TT",
             beta_estimator="hu_ok",beta_XY="TT",
             xmask=kmask,ymask=kmask,
             field_names_alpha=['E1','E2'],
             field_names_beta=['E1','E2'])
```

This modifies the total power spectra variable names that feed_dict expects. The above command will not work unless `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are also provided, instead of just the usual `tC_T_T`. Specifying these in feed_dict allows one to generalize to a wider variety of estimators.

2.1.5 2.1.5 Other built-in estimators

The following are currently available:

1. Hu Okamoto 2001 TT, TE, EE, EB, TB
2. Hu DeDeo Vale 2007 TT, TE, ET, EE, EB, TB
3. Schaan, Ferraro 2018 shear TT

For the shear estimator, the built-in variable scheme also expects duC_T_T , the logarithmic derivative of the lensed CMB temperature,

```
>>> feed_dict['duC_T_T'] =
    utils.interp(ells,np.gradient(np.log(lcltt),np.log(ells)))(modlmap)
```

Once this is added to feed_dict, noise curves and shear maps can be obtained as before,

```
>>> Nl = s.N_l(shape,wcs,feed_dict,"shear","TT",
          xmask=tmask,ymask=tmask)
>>> Al = s.A_l(shape,wcs,feed_dict,"shear","TT",xmask=tmask,ymask=tmask)
>>> ushear = s.unnormalized_quadratic_estimator(shape,wcs,feed_dict,"shear","TT",
↪xmask=tmask,ymask=tmask)
>>> shear = Al * ushear
```

2.2 2.2 Custom estimators

We can build general factorizable quadratic estimators as follows.

We need to specify the mode coupling form (little f):

$$f(\vec{l}_1, \vec{l}_2)$$

and specify the filter form (big F):

$$F(\vec{l}_1, \vec{l}_2)$$

For reference, these are related to the quadratic estimator,

$$\hat{q}(\vec{L}) = \frac{A(\vec{L})}{2} \int \frac{d^2\vec{l}_1}{(2\pi)^2} F(l_1, l_2) X(l_1) Y(l_2)$$

and normalization,

$$A(\vec{L}) = L^2 \left[\int \frac{d^2\vec{l}_1}{(2\pi)^2} F(l_1, l_2) f(l_1, l_2) \right]^{-1}$$

where $\vec{L} = \vec{l}_1 + \vec{l}_2$.

The expressions $f(\vec{l}_1, \vec{l}_2)$ and $F(\vec{l}_1, \vec{l}_2)$ must be specified in terms of the following special symbols:

1. Ldl1 for $\vec{L}.\vec{l}_1$
2. Ldl2 for $\vec{L}.\vec{l}_2$
3. cos2t12 for $\cos(2\theta_{12})$
4. sin2t12 for $\sin(2\theta_{12})$
5. L for $|\vec{L}|$

and any other arbitrary symbols which will be replaced with numerical data later on.

The special symbols can be accessed directly from the module, e.g.:

```
>>> import symlens as s
>>> s.Ldl1
>>> s.L
>>> s.cost2t12
```

and arbitrary symbols can be defined either as functions of l1 or of l2, using a wrapper in the module:

```
>>> s.e('X_l1')
>>> s.e('Y_l2')
```

The '_l1' or '_l2' suffix for arbitrary symbols is critical for the factorizer to know. With these, a large variety of estimators and noise functions can be built, including lensing, magnification, shear, birefringence, patchy tau, mixed estimators (for gradient cleaning), split lensing estimators, etc.

e.g., we can build an integrand for the Hu, Okamoto 2001 TT lensing estimator normalization as follows,

```
# Build HuOk TT estimator integrand
>>> f = s.Ldl1 * s.e('uC_T_T_l1') + s.Ldl2 * s.e('uC_T_T_l2')
>>> F = f / 2 / s.e('tC_T_T_l1') / s.e('tC_T_T_l2')
>>> expr1 = f * F # this is the integrand
```

We then provide data arrays for use after factorization in `feed_dict`. These are lensed TT spectra interpolated on to 2D Fourier space.

```
>>> feed_dict = {}
>>> feed_dict['uC_T_T'] = utils.interp(ells,ctt)(modlmap)
>>> feed_dict['tC_T_T'] = utils.interp(ells,ctt)(modlmap)
```

For the integral to be sensible, we must also mask regions in Fourier space we don't want to include.

```
>>> tellmin = 10 ; tellmax = 3000
>>> xmask = utils.mask_kspace(shape,wcs,lmin=tellmin,lmax=tellmax)
```

With these in hand, we can call the core function in symlens for the factorized integral.

```
>>> integral = s.integrate(shape,wcs,feed_dict,expr1,xmask=xmask,ymask=xmask).real
```

CHAPTER 3

3 API Reference

See *2 Usage* for how to use these functions for common tasks.

3.1 3.1 qe – Quadratic Estimators

symlens.qe.**A_l**(*shape*, *wcs*, *feed_dict*, *estimator*, *XY*, *xmask=None*, *ymask=None*, *field_names=None*, *kmask=None*)

Returns the normalization corresponding to a pre-defined mode-coupling estimator.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **estimator** (*str*) – The name of a pre-defined mode-coupling estimator. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.

- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are expected to be present in feed_dict. This allows for more custom noise correlations.

Returns **Al** – The 2D normalization for the estimator.

Return type (Ny,Nx) ndarray

symlens.qe.**A_l_custom**(*shape*, *wcs*, *feed_dict*, *f*, *F*, *xmask=None*, *ymask=None*, *groups=None*, *kmask=None*)

Returns the 2D normalization corresponding to a custom mode-coupling estimator.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (`astropy.wcs.wcs.WCS`) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **f** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the mode-coupling response. See the Usage guide for details. If this is specified, the argument F is required, and the arguments estimator, XY and field_names are ignored.
- **F** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the estimator filter. See the Usage guide for details.
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are expected to be present in feed_dict. This allows for more custom noise correlations.
- **xname** (*str,optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.
- **yname** (*str,optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.
- **groups** (*list,optional*) – Group all terms in the normalization calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **Al** – The 2D normalization for the estimator.

Return type (Ny,Nx) ndarray

symlens.qe.**N_l**(*shape*, *wcs*, *feed_dict*, *estimator*, *XY*, *xmask=None*, *ymask=None*, *Al=None*, *field_names=None*, *kmask=None*, *power_name='t'*)

Returns the 2D noise corresponding to a pre-defined mode-coupling estimator NOT assuming that it is optimal. This involves 2 integrals, unless a pre-calculated normalization Al is provided, in which case only 1 integral needs to be evaluated.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **estimator** (*str*) – The name of a pre-defined mode-coupling estimator. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **Al** (*(Ny,Nx) ndarray, optional*) – Pre-calculated normalization for the estimator. Reduces the number of integrals calculated to 1 if provided, else calculates 2 integrals.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.

Returns **Nl** – The 2D noise for the estimator.

Return type (Ny,Nx) ndarray

symlens.qe.**N_l_cross**(*shape*, *wcs*, *feed_dict*, *alpha_estimator*, *alpha_XY*, *beta_estimator*, *beta_XY*, *xmask=None*, *ymask=None*, *Aalpha=None*, *Abeta=None*, *field_names_alpha=None*, *field_names_beta=None*, *kmask=None*, *skip_filter_field_names=False*, *power_name='t'*)

Returns the 2D cross-covariance between two pre-defined mode-coupling estimators. This involves 3 integrals, unless pre-calculated normalizations Al are provided.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.

- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **alpha_estimator** (*str*) – The name of the first pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **beta_estimator** (*str, optional*) – The name of the second pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **alpha_XY** (*str, optional*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **beta_XY** (*str, optional*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **xmask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **Aalpha** (*(Ny,Nx) ndarray, optional*) – Pre-calculated normalization for the first estimator. This is calculated if not provided
- **Abeta** (*(Ny,Nx) ndarray, optional*) – Pre-calculated normalization for the second estimator. This is calculated if not provided
- **field_names_alpha** (*2 element list, optional*) – Providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are expected to be present in feed_dict. This allows for more custom noise correlations.
- **field_names_beta** (*2 element list, optional*) – As above, but for the second beta estimator.

Returns **Nl** – The requested 2D cross-covariance.

Return type (Ny,Nx) ndarray

symlens.qe.**N_l_cross_custom**(*shape*, *wcs*, *feed_dict*, *alpha_XY*, *beta_XY*, *Falpha*, *Fbeta*, *Fbeta_rev*, *xmask=None*, *ymask=None*, *field_names_alpha=None*, *field_names_beta=None*, *falpha=None*, *fbeta=None*, *Aalpha=None*, *Abeta=None*, *groups=None*, *kmask=None*, *power_name='t'*)

Returns the 2D cross-covariance between two custom mode-coupling estimators. This involves 3 integrals, unless pre-calculated normalizations Al are provided.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (`astropy.wcs.wcs.WCS`) – The wcs object completing the specification of the geometry of the footprint.

- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **alpha_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **beta_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **falpha** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the mode-coupling response for the first estimator alpha. See the Usage guide for details.
- **fbeta** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the mode-coupling response for the second estimator beta. See the Usage guide for details.
- **Falpha** (sympy.core.symbol.Symbol) – A sympy expression containing the first alpha estimator filter. See the Usage guide for details.
- **Fbeta** (sympy.core.symbol.Symbol) – A sympy expression containing the second beta estimator filter. See the Usage guide for details.
- **Fbeta_rev** (sympy.core.symbol.Symbol) – Same as above but with l1 and l2 swapped.
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names_alpha** (*2 element list, optional*) – Providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.
- **field_names_beta** (*2 element list, optional*) – As above, but for the second beta estimator.
- **groups** (*list,optional*) – Group all terms in the normalization calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.
- **Aalpha** (*(Ny,Nx) ndarray, optional*) – Pre-calculated normalization for the first estimator. This is calculated if not provided
- **Abeta** (*(Ny,Nx) ndarray, optional*) – Pre-calculated normalization for the second estimator. This is calculated if not provided

Returns **Nl** – The requested 2D cross-covariance.

Return type (Ny,Nx) ndarray

symlens.qe.**N_l_optimal**(*shape*, *wcs*, *feed_dict*, *estimator*, *XY*, *xmask=None*, *ymask=None*, *field_names=None*, *kmask=None*)

Returns the 2D noise corresponding to a pre-defined mode-coupling estimator but assuming that it is optimal, i.e. Nl = A_L L^2 / 4

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **estimator** (*str*) – The name of a pre-defined mode-coupling estimator. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.

Returns **Nl** – The 2D noise for the estimator.

Return type (Ny,Nx) ndarray

symlens.qe.**N_l_optimal_custom**(*shape*, *wcs*, *feed_dict*, *f*, *F*, *xmask=None*, *ymask=None*, *groups=None*, *kmask=None*)

Returns the 2D noise corresponding to a custom mode-coupling estimator but assuming that it is optimal, i.e. Nl = A_L L^2 / 4

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **f** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the mode-coupling response. See the Usage guide for details. If this is specified, the argument F is required, and the arguments estimator, XY and field_names are ignored.
- **F** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the estimator filter. See the Usage guide for details.

- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **xname** (*str,optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.
- **yname** (*str,optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.
- **groups** (*list,optional*) – Group all terms in the normalization calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **Nl** – The 2D noise for the estimator.

Return type (Ny,Nx) ndarray

class symlens.qe.**QE**(*shape*, *wcs*, *feed_dict*, *estimator=None*, *XY=None*, *f=None*, *F=None*, *xmask=None*, *ymask=None*, *field_names=None*, *groups=None*, *kmask=None*)

Construct a quadratic estimator such that the normalization is pre-calculated and reused.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below).
- **estimator** (*str,optional*) – The name of a pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str,optional*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **f** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the mode-coupling response. See the Usage guide for details. If this is specified, the argument F is required, and the arguments estimator, XY and field_names are ignored.
- **F** (sympy.core.symbol.Symbol , optional) – A sympy expression containing the estimator filter. See the Usage guide for details.
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.

- **groups** (*list, optional*) – Group all terms in the normalization such that they have common factors of the provided list of expressions to reduce the number of FFTs.

reconstruct (*feed_dict*, *xname='X_l1'*, *yname='Y_l2'*, *groups=None*, *physical_units=True*)

Returns a normalized reconstruction corresponding to the initialized mode-coupling estimator.

Parameters

- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **xname** (*str, optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.
- **yname** (*str, optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.
- **groups** (*list, optional*) – Group all terms in the reconstruction calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns krecon – The normalized Fourier space reconstruction in physical units (not pixel units).

Return type (Ny,Nx) ndarray

symlens.qe.**RDN0_analytic**(*shape*, *wcs*, *feed_dict*, *alpha_estimator*, *alpha_XY*, *beta_estimator*, *beta_XY*, *split_estimator=False*, *Aalpha=None*, *Abeta=None*, *xmask=None*, *ymask=None*, *kmask=None*, *field_names_alpha=None*, *field_names_beta=None*, *skip_filter_field_names=False*)

Often lovingly called the 'dumb' N0 by ACT lensers, this is the analytic expression for the realization-dependependent N0 correction when the noise is isotropic and no mask is present.

feed_dict should have dC_T_T, etc. the realized total data power spectrum. tC_T_T, etc. should be the total coadd power spectrum used in filters uC_T_T, etc. the usual theory spectra for the CMB signal. nC_T_T etc. should be the expected total theory power spectrum But if split_estimator is true: dC_T_T, etc. should be the realized cross-power average. nC_T_T etc. should be the expected cross-power, which is usually nC without the instrument noise.

symlens.qe.**RDN0_analytic_generic**(*shape*, *wcs*, *feed_dict*, *alpha_XY*, *beta_XY*, *Falpha*, *Fbeta*, *Fbeta_rev*, *falpha=None*, *fbeta=None*, *Aalpha=None*, *Abeta=None*, *xmask=None*, *ymask=None*, *kmask=None*, *field_names_alpha=None*, *field_names_beta=None*, *split_estimator=False*, *groups=None*)

Often lovingly called the 'dumb' N0 by ACT lensers, this is the analytic expression for the realization-dependependent N0 correction when the noise is isotropic and no mask is present.

feed_dict should have dC_T_T, etc. the realized total data power spectrum. tC_T_T, etc. should be the total coadd power spectrum used in filters uC_T_T, etc. the usual theory spectra for the CMB signal. nC_T_T etc. should be the expected total theory power spectrum But if split_estimator is true: dC_T_T, etc. should be the realized cross-power average. nC_T_T etc. should be the expected cross-power, which is usually nC without the instrument noise.

symlens.qe.**cross_integral_custom**(*shape*, *wcs*, *feed_dict*, *alpha_XY*, *beta_XY*, *Falpha*, *Fbeta*, *Fbeta_rev*, *xmask=None*, *ymask=None*, *field_names_alpha=None*, *field_names_beta=None*, *groups=None*, *power_name='t'*)

Calculates the integral

$$\int \frac{d^2\vec{l}_1}{(2\pi)^2} F_\alpha(\vec{l}_1,\vec{l}_2)(F_\beta(\vec{l}_1,\vec{l}_2)C^{ac}_{l_1}C^{bd}_{l_2} + F_\beta(\vec{l}_2,\vec{l}_1)C^{ad}_{l_1}C^{bc}_{l_2})$$

where alpha_XY = "ab" beta_XY = "cd"

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **alpha_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **beta_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **Falpha** (sympy.core.symbol.Symbol) – A sympy expression containing the first alpha estimator filter. See the Usage guide for details.
- **Fbeta** (sympy.core.symbol.Symbol) – A sympy expression containing the second beta estimator filter. See the Usage guide for details.
- **Fbeta_rev** (sympy.core.symbol.Symbol) – Same as above but with l1 and l2 swapped.
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names_alpha** (*2 element list, optional*) – Providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.
- **field_names_beta** (*2 element list, optional*) – As above, but for the second beta estimator.
- **groups** (*list,optional*) – Group all terms in the normalization calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **integral** – Returns the integral described above.

Return type (Ny,Nx) ndarray

symlens.qe.**generic_cross_integral**(*shape*, *wcs*, *feed_dict*, *alpha_XY*, *beta_XY*, *Falpha*, *Fbeta*, *Fbeta_rev*, *Dexpr1*, *Dexpr2*, *xmask=None*, *ymask=None*, *field_names_alpha=None*, *field_names_beta=None*, *groups=None*)

Calculates the integral

$$\int \frac{d^2\vec{l}_1}{(2\pi)^2} F_\alpha(\vec{l}_1,\vec{l}_2)(F_\beta(\vec{l}_1,\vec{l}_2)D_1(l_1,l_2)+F_\beta(\vec{l}_2,\vec{l}_1)D_2(l_1,l_2))$$

where alpha_XY = "ab" beta_XY = "cd"

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **alpha_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **beta_XY** (*str*) – The XY pair for the first estimator. Typical examples include "TT" and "EB".
- **Falpha** (sympy.core.symbol.Symbol) – A sympy expression containing the first alpha estimator filter. See the Usage guide for details.
- **Fbeta** (sympy.core.symbol.Symbol) – A sympy expression containing the second beta estimator filter. See the Usage guide for details.
- **Fbeta_rev** (sympy.core.symbol.Symbol) – Same as above but with l1 and l2 swapped.
- **Dexpr1** (sympy.core.symbol.Symbol) – A sympy expression entering in the generic integral.
- **Dexpr2** (sympy.core.symbol.Symbol) – A second sympy expression entering in the generic integral.
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names_alpha** (*2 element list, optional*) – Providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.
- **field_names_beta** (*2 element list, optional*) – As above, but for the second beta estimator.

- **groups** (*list, optional*) – Group all terms in the normalization calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **integral** – Returns the integral described above.

Return type (Ny,Nx) ndarray

symlens.qe.**generic_noise_expression**(*cross_integral*, *shape*, *wcs*, *feed_dict*, *falpha*, *fbeta*, *Falpha*, *Fbeta*, *xmask=None*, *ymask=None*, *kmask=None*, *Aalpha=None*, *Abeta=None*)

returns (1/4) A_alpha * A_beta * cross_integral

symlens.qe.**get_mc_expressions**(*estimator*, *XY*, *field_names=None*, *estimator_to_harden='hu_ok'*)

Pre-defined mode coupling expressions. Returns f(l1,l2), F(l1,l2), F(l2,l1). If a list field_names is provided containing two strings, then "total power" spectra are customized to potentially be different and feed_dict will need to have more values.

Parameters

- **estimator** (*str*) – The name of a pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are expected to be present in feed_dict. This allows for more general noise correlations.

Returns

- **f** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the mode-coupling response. See the Usage guide for details.
- **F** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the estimator filter.
- **Fr** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the estimator filter but with l1 and l2 swapped.

symlens.qe.**lensing_response_f**(*XY*, *rev=False*, *curl=False*)

Returns the mode-coupling response f(l1,l2) for CMB lensing.

Parameters

- **XY** (*str*) – The XY pair for the requested estimator. This must belong to one of TT, EE, TE, ET, EB or TB.
- **rev** (*boolean, optional*) – Whether to swap l1 and l2. Defaults to False.

Returns **f** – A sympy expression containing the mode-coupling response. See the Usage guide for details.

Return type `sympy.core.symbol.Symbol` , optional

symlens.qe.**reconstruct**(*shape*, *wcs*, *feed_dict*, *estimator=None*, *XY=None*, *f=None*, *F=None*, *xmask=None*, *ymask=None*, *field_names=None*, *norm_groups=None*, *est_groups=None*, *xname='X_l1'*, *yname='Y_l2'*, *kmask=None*, *physical_units=True*)

Returns a normalized reconstruction corresponding to specified mode-coupling estimator.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (`astropy.wcs.wcs.WCS`) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **estimator** (*str, optional*) – The name of a pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str, optional*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **f** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the mode-coupling response. See the Usage guide for details. If this is specified, the argument F is required, and the arguments estimator, XY and field_names are ignored.
- **F** (`sympy.core.symbol.Symbol` , optional) – A sympy expression containing the estimator filter. See the Usage guide for details.
- **xmask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like `tC_E1_T_E1_T`, `tC_E2_T_E2_T`, `tC_E1_T_E2_T`, `tC_E2_T_E1_T` are expected to be present in feed_dict. This allows for more custom noise correlations.
- **norm_groups** (*list, optional*) – Group all terms in the normalization such that they have common factors of the provided list of expressions to reduce the number of FFTs.
- **xname** (*str, optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.
- **yname** (*str, optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.
- **est_groups** (*list, optional*) – Group all terms in the reconstruction calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **krecon** – The normalized Fourier space reconstruction in physical units (not pixel units).

Return type (Ny,Nx) ndarray

See also:

QE() A class with which the slow normalization can be pre-calculated and repeated estimation can be performed on similar datasets.

symlens.qe.**rotation_response_f**(*XY*, *rev=False*)

Returns the mode-coupling response f(l1,l2) for CMB rotation.

Parameters

- **XY** (*str*) – The XY pair for the requested estimator. This must belong to one of EE, TE, ET, EB or TB.
- **rev** (*boolean, optional*) – Whether to swap l1 and l2. Defaults to False.

Returns **f** – A sympy expression containing the mode-coupling response. See the Usage guide for details.

Return type sympy.core.symbol.Symbol

symlens.qe.**unnormalized_quadratic_estimator**(*shape*, *wcs*, *feed_dict*, *estimator*, *XY*, *xname='X_l1'*, *yname='Y_l2'*, *field_names=None*, *xmask=None*, *ymask=None*, *physical_units=True*)

Returns a normalized reconstruction corresponding to specified pre-defined mode-coupling estimator.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **estimator** (*str,optional*) – The name of a pre-defined mode-coupling estimator. If this is provided, the argument XY is required and the arguments f, F, and groups are ignored. e.g. "hu_ok", "hdv" and "shear".
- **XY** (*str,optional*) – The XY pair for the requested estimator. Typical examples include "TT" and "EB".
- **xmask** (*(Ny,Nx) ndarray,optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **field_names** (*2 element list, optional*) – When a pre-defined mode-coupling estimator is used, providing a list field_names modifies the total power spectra variable names that feed_dict expects. Typically, names like "tC_T_T" and "tC_T_E" are expected. But if field_names is ["E1","E2"] for example, variable names like tC_E1_T_E1_T, tC_E2_T_E2_T, tC_E1_T_E2_T, tC_E2_T_E1_T are expected to be present in feed_dict. This allows for more custom noise correlations.
- **xname** (*str,optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.

- **yname** (*str, optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.

Returns **krecon** – The normalized Fourier space reconstruction in physical units (not pixel units).

Return type (Ny,Nx) ndarray

See also:

reconstruct() Get the properly normalized quadratic estimator reconstruction.

QE() A class with which the slow normalization can be pre-calculated and repeated estimation can be performed on similar datasets.

symlens.qe.**unnormalized_quadratic_estimator_custom**(*shape*, *wcs*, *feed_dict*, *F*, *xname='X_l1'*, *yname='Y_l2'*, *xmask=None*, *ymask=None*, *groups=None*, *physical_units=True*)

Returns a normalized reconstruction corresponding to a custom mode-coupling estimator.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (astropy.wcs.wcs.WCS) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays used in the normalization and reconstruction calculation. When using pre-defined mode-coupling estimators, typical keys that must be present are 'uC_X_Y' and 'tC_X_Y', where X and Y depend on the requested estimator XY (see below). This feed_dict must also contain the keys with name xname and yname (see below), which contain the 2D maps X and Y for the data from which the quadratic estimate is constructed.
- **F** (sympy.core.symbol.Symbol) – A sympy expression containing the estimator filter. See the Usage guide for details.
- **xmask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **xname** (*str, optional*) – The name of the key in feed_dict where the X map in the XY quadratic estimator is stored. Defaults to X_l1.
- **yname** (*str, optional*) – The name of the key in feed_dict where the Y map in the XY quadratic estimator is stored. Defaults to Y_l2.
- **groups** (*list, optional*) – Group all terms in the reconstruction calclulation such that they have common factors of the provided list of expressions to reduce the number of FFTs.

Returns **krecon** – The normalized Fourier space reconstruction in physical units (not pixel units).

Return type (Ny,Nx) ndarray

See also:

QE() A class with which the slow normalization can be pre-calculated and repeated estimation can be performed on similar datasets.

3.2 3.2 factorize – Core utilties for factorization and integration

symlens.factorize.**factorize_2d_convolution_integral**(*expr*, *l1funcs=None*, *l2funcs=None*, *groups=None*, *validate=True*)

Reduce a sympy expression of variables l1x,l1y,l2x,l2y,l1,l2 into a sum of products of factors that depend only on vec(l1) and vec(l2) and neither, each. If the expression appeared as the integrand in an integral over vec(l1), where vec(l2) = vec(L) - vec(l1) then this reduction allows one to evaluate the integral as a function of vec(L) using FFTs instead of as a convolution.

Parameters

- **expr** (`sympy.core.symbol.Symbol`) – The full Sympy expression to reduce to sum of products of functions of l1 and l2.
- **l1funcs** (*list*) – List of symbols that are functions of l1
- **l2funcs** (*list*) – List of symbols that are functions of l2
- **groups** (*list, optional*) – Group all terms such that they have common factors of the provided list of expressions to reduce the number of FFTs.
- **validate** (*boolean, optional*) – Whether to check that the final expression and the original agree. Defaults to True.

Returns

- *terms*
- *unique_l1s*
- *unique_l2s*
- *ogroups*
- *ogroup_weights*
- *ogroup_symbols*

symlens.factorize.**integrate**(*shape*, *wcs*, *feed_dict*, *expr*, *xmask=None*, *ymask=None*, *cache=True*, *validate=True*, *groups=None*, *physical_units=True*)

Integrate an arbitrary expression after factorizing it.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (`astropy.wcs.wcs.WCS`) – The wcs object completing the specification of the geometry of the footprint.
- **feed_dict** (*dict*) – Mapping from names of custom symbols to numpy arrays.
- **expr** (`sympy.core.symbol.Symbol`) – A sympy expression containing recognized symbols (see docs)
- **xmask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l1 part of the integral. Defaults to ones.
- **ymask** (*(Ny,Nx) ndarray, optional*) – Fourier space 2D mask for the l2 part of the integral. Defaults to ones.
- **cache** (*boolean, optional*) – Whether to store in memory and reuse repeated terms. Defaults to true.

- **validate** (*boolean, optional*) – Whether to check that the final expression and the original agree. Defaults to True.
- **groups** (*list, optional*) – Group all terms such that they have common factors of the provided list of expressions to reduce the number of FFTs.
- **physical_units** (*boolean, optional*) – Whether the input is in pixel units or not.

Returns **result** – The numerical result of the integration of the expression after factorization.

Return type (Ny,Nx) ndarray

3.3 3.3 utils - General utilities

symlens.utils.**evaluate**(*symbolic_term*, *feed_dict*)

Convert a symbolic term into a numerical result by using values for the symbols from a dictionary.

symbolic_term: sympy expression feed_dict: dictionary mapping names of symbols to numpy arrays

symlens.utils.**gauss_beam**(*ells*, *fwhm*)

Return a Gaussian beam transfer function for the given ells.

Parameters

- **ells** (*ndarray*) – Any numpy array containing the multipoles at which the beam transfer function is requested.
- **fwhm** (*float*) – The beam FWHM in arcminutes.

Returns **output** – An array of the same shape as ells containing the Gaussian beam transfer function for those multipoles.

Return type ndarray

symlens.utils.**interp**(*x*, *y*, *bounds_error=False*, *fill_value=0.0*, ***kwargs*)

Return a function that interpolates (x,y). This wraps around scipy.interpolate.interp1d but by defaulting to zero filling outside bounds.

Docstring copied from scipy. Interpolate a 1-D function. *x* and *y* are arrays of values used to approximate some function f: `y = f(x)`. This class returns a function whose call method uses interpolation to find the value of new points. Note that calling *interp1d* with NaNs present in input values results in undefined behaviour. :param x: A 1-D array of real values. :type x: (N,) array_like :param y: A N-D array of real values. The length of *y* along the interpolation

axis must be equal to the length of *x*.

Parameters

- **kind** (*str or int, optional*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'previous', 'next', where 'zero', 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of zeroth, first, second or third order; 'previous' and 'next' simply return the previous or next value of the point) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.
- **axis** (*int, optional*) – Specifies the axis of *y* along which to interpolate. Interpolation defaults to the last axis of *y*.
- **copy** (*bool, optional*) – If True, the class makes internal copies of x and y. If False, references to *x* and *y* are used. The default is to copy.

- **bounds_error** (*bool, optional*) – If True, a ValueError is raised any time interpolation is attempted on a value outside of the range of x (where extrapolation is necessary). If False, out of bounds values are assigned *fill_value*. False by default.
- **fill_value** (*array-like or (array-like, array_like) or "extrapolate", optional*) –
  - if a ndarray (or float), this value will be used to fill in for requested points outside of the data range. If not provided, then the default is zero. The array-like must broadcast properly to the dimensions of the non-interpolation axes.
  - If a two-element tuple, then the first element is used as a fill value for `x_new < x[0]` and the second element is used for `x_new > x[-1]`. Anything that is not a 2-element tuple (e.g., list or ndarray, regardless of shape) is taken to be a single array-like argument meant to be used for both bounds as `below, above = fill_value, fill_value`. .. versionadded:: 0.17.0
  - If "extrapolate", then points outside the data range will be extrapolated. .. versionadded:: 0.17.0
- **assume_sorted** (*bool, optional*) – If False, values of *x* can be in any order and they are sorted first. If True, *x* has to be an array of monotonically increasing values.

`symlens.utils.__call__()`

See also:

`splrep()`, `splev()`

`UnivariateSpline()` An object-oriented wrapper of the FITPACK routines.

`interp2d()` 2-D interpolation

Examples

```
>>> import matplotlib.pyplot as plt
>>> from scipy import interpolate
>>> x = np.arange(0, 10)
>>> y = np.exp(-x/3.0)
>>> f = interpolate.interp1d(x, y)
>>> xnew = np.arange(0, 9, 0.1)
>>> ynew = f(xnew)   # use interpolation function returned by `interp1d`
>>> plt.plot(x, y, 'o', xnew, ynew, '-')
>>> plt.show()
```

`symlens.utils.`**`mask_kspace`**(*shape*, *wcs*, *lxcut=None*, *lycut=None*, *lmin=None*, *lmax=None*)

Produce a Fourier space mask.

Parameters

- **shape** (*tuple*) – The shape of the array for the geometry of the footprint. Typically (...,Ny,Nx) for Ny pixels in the y-direction and Nx in the x-direction.
- **wcs** (`astropy.wcs.wcs.WCS`) – The wcs object completing the specification of the geometry of the footprint.
- **lxcut** (*int, optional*) – The width of a band in number of Fourier pixels to be masked in the lx direction. Default is no masking in this band.
- **lycut** (*int, optional*) – The width of a band in number of Fourier pixels to be masked in the ly direction. Default is no masking in this band.

- **lmin** (*int, optional*) – The radial distance in Fourier space below which all Fourier pixels are masked. Default is no masking.
- **lmax** (*int, optional*) – The radial distance in Fourier space above which all Fourier pixels are masked. Default is no masking.

Returns **output** – A 2D array containing the Fourier space mask.

Return type (Ny,Nx) ndarray

symlens.utils.**rect_geometry**(*width_arcmin=None*, *width_deg=None*, *px_res_arcmin=0.5*, *proj='car'*, *pol=False*, *height_deg=None*, *height_arcmin=None*, *xoffset_degree=0.0*, *yoffset_degree=0.0*, *extra=False*, ***kwargs*)

Get shape and wcs for a rectangular patch of specified size and coordinate center

CHAPTER 4

4 History

4.1 4.1 0.1.0 (2019-03-06)

- First release on PyPI.

4.2 4.2 0.3.3 (2020-07-25)

- Better treatment of FFT factors
- Lots of new features including bias hardening and isotropic RDN0

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

C

E

F

G

I

L

M

N

Q

R

S

U