
schnell Documentation

Release 0.2.0

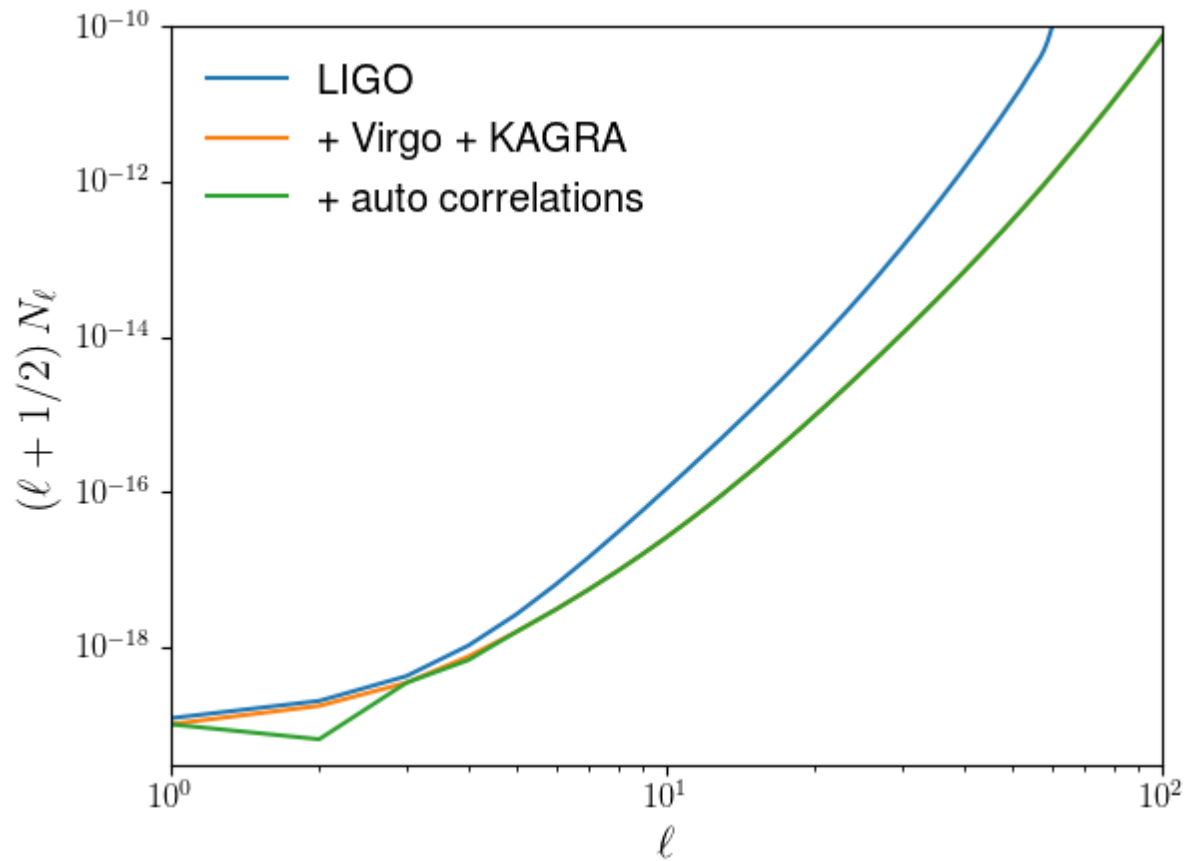
David Alonso

May 03, 2020

Contents:

1	Overview	1
1.1	Installation	2
2	API Documentation	3
2.1	Detectors	3
2.2	Correlations	7
2.3	Map calculators	8
3	Indices and tables	13
	Python Module Index	15
	Index	17

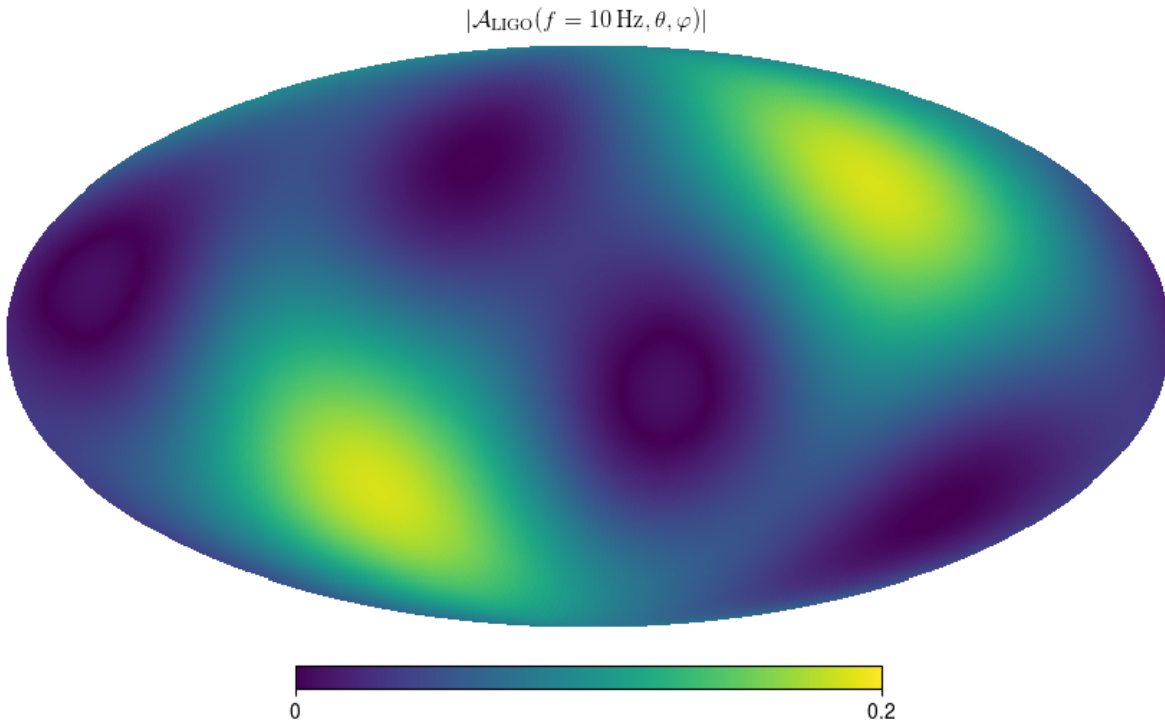
schNell is a very lightweight python module that can be used to compute basic map-level noise properties for generic networks of gravitational wave interferometers. This includes primarily the noise power spectrum N_ℓ , but also other things, such as antenna patterns, overlap functions, inverse variance maps etc.



schNell is composed of three main classes:

- **Detectors.** These contain information about each individual detector of the network (their positions, noise properties, orientation etc.).
- **NoiseCorrelations.** These describe the noise-level correlation between pairs of detectors.
- **MapCalculators.** These objects combine a list of *Detectors* into a network (potentially together with a *NoiseCorrelation* object) and compute the corresponding map-level noise properties arising from their correlations.

A quick but thorough description of how these three classes can be used to compute different quantities can be found in [here](#).



1.1 Installation

Installing *schNell* should be as simple as typing:

```
pip install schnell [--user]
```

where the `--user` flag will only be necessary if you don't have admin privileges.

The three classes that make up *schNell* are described here. Please refer to the examples on github for a description of how these interact in practice.

2.1 Detectors

class `schnell.detector.Detector` (*name*)

Detector objects encode information about individual GW detectors. The most relevant quantities are:

- Detector position.
- Detector transfer function.
- Unit vectors in arm directions.
- Detector response tensor.
- Noise PSDs.

Baseline *Detector* objects serve only as a superclass for all other detector types. Do not use them.

get_Fp (*t, f, e_p, e_x, nv*)

Compute the quantity:

$$F^p(f, \hat{n}) = a^{ij} e_{ij}^p \exp(i2\pi f \hat{n} \mathbf{x})$$

(i.e. Eq. 12 of the companion paper).

Parameters

- **t** – array of size N_t containing observing times (in s).
- **f** – array of size N_f containing frequencies (in Hz).
- **ep** – array of shape $[3, 3, N_{pix}]$ containing the “+” polarization tensor at N_{pix} different sky positions.
- **ex** – same as *ep* for the “x” polarization tensor.

- **nv** – array of shape $[3, N_{pix}]$ containing the unit vector pointing in the direction of N_{pix} sky positions.

Returns 2 arrays of shape $[N_t, N_f, N_{pix}]$ containing F^+ and $F^\times$ as a function of time, frequency and sky position.

Return type array_like

get_position (t)

Returns a 2D array containing the 3D position of the detector at a series of times. The output array has shape $[3, N_t]$, where N_t is the size of t .

Parameters **t** – time of observation (in seconds).

Returns detector position (in m) as a function of time.

Return type array_like

get_transfer (u, f, nv)

Returns the detector transfer function as a function of position, frequency and sky coordinates.

Parameters

- **u** – 2D array of size $[3, N_t]$ containing the unit vector pointing along one of the detector arms at N_t different time intervals.
- **f** – 1D array of frequencies (in Hz).
- **nv** – 2D array of shape $[3, N_{pix}]$ containing the normalized coordinates of N_{pix} points in the celestial sphere.

Returns array of shape $[N_t, N_f, N_{pix}]$ containing the transfer function as a function of time, frequency and sky position.

Return type array_like

get_uu_vv (t)

Returns the outer product of the detector arm unit vectors as a function of time.

Parameters **t** – array of size N_t containing different observing times (in s).

Returns 2 arrays of shape $[3, 3, N_t]$ containing the outer products of the unit vectors pointing in the directions of the two detector arms.

Return type array_like

psd (f)

Returns noise PSD as a function of frequency.

Parameters **f** – array of frequencies (in Hz).

Returns array of PSD values in units of 1/Hz.

Return type array_like

class schnell.detector.**GroundDetector** ($name, lat, lon, alpha, fname_psd, aperture=90$)

GroundDetector objects represent detectors located at fixed position on Earth.

Parameters

- **name** – detector name.
- **lat** – latitude in degrees.
- **lon** – longitude in degrees.

- **alpha** – orientation angle, defined as the angle between the vertex bisector and the local parallel. In degrees.
- **fname_psd** – path to file containing the detector’s noise curve. The file should contain two columns, corresponding to the frequency (in Hz) and the corresponding value of the strain-level noise (in units of $\text{Hz}^{-1/2}$).
- **aperture** – arm aperture angle (in degrees).

get_position(*t*)

Returns a 2D array containing the 3D position of the detector at a series of times. The output array has shape $[3, N_t]$, where N_t is the size of *t*.

Note: We assume the Earth is a sphere of radius 6371 km that performs a full rotation every 24 h exactly.

Parameters *t* – time of observation (in seconds).

Returns detector position (in m) as a function of time.

Return type array_like

get_u_v(*t*)

Returns unit vectors in the directions of the detector arms as a function of time.

Parameters *t* – array of size N_t containing different observing times (in s).

Returns 2 arrays of shape $[3, N_t]$ containing the arm unit vectors.

Return type array_like

class schnell.detector.**GroundDetectorTriangle**(*name, lat, lon, fname_psd, detector_id, beta0_deg=0, arm_length_km=10.0*)

GroundDetectorTriangle objects represent detectors in a triangular configuration located at fixed position on Earth (e.g. the Einstein Telescope).

Parameters

- **name** – detector name.
- **lat** – latitude of the triangle’s barycenter in degrees.
- **lon** – longitude in degrees.
- **fname_psd** – path to file containing the detector’s noise curve. The file should contain two columns, corresponding to the frequency (in Hz) and the corresponding value of the strain-level noise (in units of $\text{Hz}^{-1/2}$).
- **detector_id** – detector number (0, 1 or 2).
- **beta0_deg** – orientation angle, defined as the angle between the vertex with id 0 and the local meridian.
- **arm_length_km** – arm length in km.

get_position(*t*)

Returns a 2D array containing the 3D position of the detector at a series of times. The output array has shape $[3, N_t]$, where N_t is the size of *t*.

Note: We assume the Earth is a sphere of radius 6371 km that performs a full rotation every 24 h exactly.

Parameters t – time of observation (in seconds).

Returns detector position (in m) as a function of time.

Return type array_like

get_u_v(t)

Returns unit vectors in the directions of the detector arms as a function of time.

Parameters t – array of size N_t containing different observing times (in s).

Returns 2 arrays of shape $[3, N_t]$ containing the arm unit vectors.

Return type array_like

class schnell.detector.LISADetector(*detector_id, is_L5Gm=False, static=False*)

LISADetector objects can be used to describe the properties of the LISA network.

Parameters

- **detector_id** – detector number (0, 1 or 2).
- **is_L5Gm** (*bool*) – whether the arm length should be 5E9 meters (otherwise 2.5E9 meters will be assumed) (default *False*).
- **static** (*bool*) – if *True*, a static configuration corresponding to a perfect equilateral triangle in the x-y plane will be assumed.

get_position(t)

Returns a 2D array containing the 3D position of the detector at a series of times. The output array has shape $[3, N_t]$, where N_t is the size of t .

Note: The spacecraft orbits are calculated using Eq. 1 of gr-qc/0311069.

Parameters t – time of observation (in seconds).

Returns detector position (in m) as a function of time.

Return type array_like

get_u_v(t)

Returns unit vectors in the directions of the detector arms as a function of time.

Parameters t – array of size N_t containing different observing times (in s).

Returns 2 arrays of shape $[3, N_t]$ containing the arm unit vectors.

Return type array_like

psd(f)

Returns noise PSD as a function of frequency. Uses Eq. 55 from arXiv:1908.00546.

Parameters f – array of frequencies (in Hz).

Returns array of PSD values in units of 1/Hz.

Return type array_like

psd_A(f)

Returns auto-noise PSD as a function of frequency. Uses Eq. 55 from arXiv:1908.00546.

Parameters f – array of frequencies (in Hz).

Returns array of PSD values in units of 1/Hz.

Return type array_like

psd_x(*f*)

Returns cross-noise PSD as a function of frequency. Uses Eq. 56 from arXiv:1908.00546.

Parameters *f* – array of frequencies (in Hz).

Returns array of PSD values in units of 1/Hz.

Return type array_like

2.2 Correlations

class schnell.correlation.NoiseCorrelationBase(*ndet*)

Noise correlation objects have methods to compute noise PSD correlation matrices.

Do not use the bare class.

get_corrmat(*f*)

Return covariance matrix as a function of frequency.

Parameters *f* – array of N_f frequencies.

Returns array of shape $[N_f, N_d, N_d]$, where N_d is the number of detectors in the network, containing the correlation matrix for each input frequency.

Return type array_like

class schnell.correlation.NoiseCorrelationConstant(*corrmat*)

This describes constant correlation matrices.

Parameters *corrmat* – 2D array providing the constant covariance matrix.

class schnell.correlation.NoiseCorrelationConstantIdentity(*ndet*)

This describes diagonal correlation matrices.

Parameters *ndet* – number of detectors in the network.

class schnell.correlation.NoiseCorrelationConstantR(*ndet*, *r*)

This class implements correlation matrices that have the same cross-correlation coefficient for all pairs of different detector, which is also constant in frequency.

Parameters

- *ndet* – number of detectors in the network.
- *r* – pairwise correlation coefficient.

class schnell.correlation.NoiseCorrelationFromFunctions(*ndet*, *psd_auto*, *psd_cross*)

This implements a correlation matrix that has the same auto-correlation PSD for all detectors and the same cross-correlation PSD for all pairs of different detectors.

Parameters

- *ndet* – number of detectors in the network.
- *psd_auto* – function of frequency returning the detector noise auto-correlation.
- *psd_cross* – function of frequency returning the detector noise cross-correlation.

class schnell.correlation.NoiseCorrelationLISA(*det*)

This implements the LISA noise correlation matrix.

Parameters *det* – LISADetector object.

2.3 Map calculators

```
class schnell.mapping.MapCalculator (det_array, f_pivot=63.0, spectral_index=0.6666666666666666, corr_matrix=None, h=0.67)
```

Map calculators compute map-level quantities for a given network of detectors.

Parameters

- **det_array** – list of *Detector* objects.
- **f_pivot** – pivot frequency in Hz (default: 63 Hz)
- **spectral_index** – power-law spectral index. This should correspond to the index in units of Ω_{GW} , not intensity.
- **corr_matrix** – noise correlation matrix for the array. If *None* the identity is assumed. If a constant, this will be assumed to be the correlation coefficient between pairs of different detectors. If a 2D array, it will be the frequency-independent correlation matrix. Otherwise, pass a *NoiseCorrelationBase* object.
- **h** – value of the Hubble constant in units of 100 km/s/Mpc (default: 0.67).

```
get_G_ell (t, f, nside, no_autos=False, deltaOmega_norm=True, proj=None)
```

Computes G_ℓ in Eq. 37 of the companion paper.

Parameters

- **t** – array of N_t time values (in s).
- **f** – array of N_f frequency values (in Hz).
- **nside** – HEALPix resolution parameter used to compute spherical harmonic transforms.
- **no_autos** (*bool*, or *array_like*) – if a single *True* value, all detector auto-correlations will be removed. If a 1D array, only the auto-correlations for which the array element is *True* will be removed. If a 2D array, all autos and cross-correlations for which the array element is *True* will be removed.
- **deltaOmega_norm** – if *True*, the quantity being mapped is $\delta\Omega = (\Omega/\bar{\Omega} - 1)/4\pi$. Otherwise the 4π factor is omitted. (Default: *True*).
- **proj** (dictionary or *None*) – if you want to project the data onto a set of linear combinations of the detectors, pass the linear coefficients of those here. *proj* should be a dictionary with two items: ‘vectors’ containing a 2D array (or a single vector) with the linear coefficients as rows and ‘deproj’. If ‘deproj’ is *True*, then those linear combinations will actually be projected out. If *proj* is *None*, then no projection or de-projection will happen.

Returns

array of shape $[N_f, N_t, N_l]$, where $N_l = 3 * nside$, containing G_ℓ at each frequency and time.

Return type *array_like*

```
get_N_ell (t, f, nside, is_fspacing_log=False, no_autos=False, deltaOmega_norm=True, proj=None)
```

Computes N_ℓ for this network.

Parameters

- **t** (*float* or *array_like*) – N_t time values (in s). If a single number is passed, then the “rigid network” approximation is used, and this time is interpreted as the total observing time. Otherwise, an integral over time is performed.

- **f** – array of N_f frequency values (in Hz).
- **nside** – HEALPix resolution parameter used to compute spherical harmonic transforms.
- **is_fspacing_log** – if *True*, f is log-spaced (linearly-spaced otherwise). (Default: *False*).
- **no_autos** (*bool*, or *array_like*) – if a single *True* value, all detector auto-correlations will be removed. If a 1D array, only the auto-correlations for which the array element is *True* will be removed. If a 2D array, all autos and cross- correlations for which the array element is *True* will be removed.
- **deltaOmega_norm** – if *True*, the quantity being mapped is $\delta\Omega = (\Omega/\bar{\Omega} - 1)/4\pi$. Otherwise the 4π factor is omitted. (Default: *True*).
- **proj** (dictionary or *None*) – if you want to project the data onto a set of linear combinations of the detectors, pass the linear coefficients of those here. *proj* should be a dictionary with two items: ‘vectors’ containing a 2D array (or a single vector) with the linear coefficients as rows and ‘deproj’. If ‘deproj’ is *True*, then those linear combinations will actually be projected out. If *proj* is *None*, then no projection or de-projection will happen.

Returns

array of size $N_l = 3 * nside$ containing the noise power spectrum.

Return type array_like

get_Ninv_t (*t*, *f*, *nside*, *is_fspacing_log*=*False*, *no_autos*=*False*, *deltaOmega_norm*=*True*, *proj*=*None*)

Computes inverse noise variance map for a set of timeframes integrated over frequency.

Parameters

- **t** – array of N_t time values (in s).
- **f** – array of frequency values that will be integrated over.
- **nside** – HEALPix resolution parameter.
- **is_fspacing_log** – if *True*, f is log-spaced (linearly-spaced otherwise). (Default: *False*).
- **no_autos** (*bool*, or *array_like*) – if a single *True* value, all detector auto-correlations will be removed. If a 1D array, only the auto-correlations for which the array element is *True* will be removed. If a 2D array, all autos and cross- correlations for which the array element is *True* will be removed.
- **deltaOmega_norm** – if *True*, the quantity being mapped is $\delta\Omega = (\Omega/\bar{\Omega} - 1)/4\pi$. Otherwise the 4π factor is omitted. (Default: *True*).
- **proj** (dictionary or *None*) – if you want to project the data onto a set of linear combinations of the detectors, pass the linear coefficients of those here. *proj* should be a dictionary with two items: ‘vectors’ containing a 2D array (or a single vector) with the linear coefficients as rows and ‘deproj’. If ‘deproj’ is *True*, then those linear combinations will actually be projected out. If *proj* is *None*, then no projection or de-projection will happen.

Returns array of shape $[N_t, N_{pix}]$ containing the inverse noise variance map sampled at the N_{pix} pixel positions corresponding to the input HEALPix resolution parameter (in RING ordering).

Return type array_like

get_antenna (*i*, *j*, *t*, *f*, *theta*, *phi*, *pol*=*False*, *inc_baseline*=*True*)

Returns antenna pattern for a detector pair as a function of time, frequency and sky position.

Parameters

- **i** – index of first detector
- **j** – index of second detector
- **t** – array of N_t times (in s).
- **f** – array of N_f times (in Hz).
- **theta** – array of N_{pix} colatitude values (in radians).
- **phi** – array of N_{pix} azimuth values (in radians).
- **pol** (*bool*) – compute all polarized components? (default: False).
- **inc_baseline** – include baseline-related phase. Otherwise only the γ overlap function in Eq. 22 of the companion paper will be returned. (default: True).

get_dsigm2_dnu_t (*t, f, nside, no_autos=False, proj=None*)

Computes $d\sigma^{-2}/df dt$ for a set of frequencies and times.

Parameters

- **t** – array of N_t time values (in s).
- **f** – array of N_f frequency values (in Hz).
- **nside** – HEALPix resolution parameter. Used to create maps of the antenna pattern and computes its sky average.
- **no_autos** (*bool, or array_like*) – if a single *True* value, all detector auto-correlations will be removed. If a 1D array, only the auto-correlations for which the array element is *True* will be removed. If a 2D array, all autos and cross- correlations for which the array element is *True* will be removed.
- **proj** (*dictionary or None*) – if you want to project the data onto a set of linear combinations of the detectors, pass the linear coefficients of those here. *proj* should be a dictionary with two items: ‘vectors’ containing a 2D array (or a single vector) with the linear coefficients as rows and ‘deproj’. If ‘deproj’ is *True*, then those linear combinations will actually be projected out. If *proj* is *None*, then no projection or de-projection will happen.

Returns array of shape $[N_t, N_f]$.

Return type array_like

get_pi_curve (*t, f, nside, is_fspacing_log=False, no_autos=False, beta_range=[-10, 10], nsigma=1, proj=None*)

Computes the power-law-integrated (PI) sensitivity curve for this network (see arXiv:1310.5300).

Parameters

- **t** (*float or array_like*) – N_t time values (in s). If a single number is passed, then the “rigid network” approximation is used, and this time is interpreted as the total observing time. Otherwise, an integral over time is performed.
- **f** – array of N_f frequency values (in Hz). This will be the frequencies at which the PI curve will be sampled, and also the frequencies used for numerical integration.
- **nside** – HEALPix resolution parameter. Used to create maps of the antenna pattern and computes its sky average.
- **no_autos** (*bool, or array_like*) – if a single *True* value, all detector auto-correlations will be removed. If a 1D array, only the auto-correlations for which the array element is *True* will be removed. If a 2D array, all autos and cross- correlations for which the array element is *True* will be removed.

- **beta_range** – a list containing the range of power law indices for which the PI curve will be computed.
- **nsigma** – S/N of the PI curve (default: 1-sigma).
- **proj** (dictionary or *None*) – if you want to project the data onto a set of linear combinations of the detectors, pass the linear coefficients of those here. *proj* should be a dictionary with two items: ‘vectors’ containing a 2D array (or a single vector) with the linear coefficients as rows and ‘deproj’. If ‘deproj’ is *True*, then those linear combinations will actually be projected out. If *proj* is *None*, then no projection or de-projection will happen.

Returns array of size N_f .

Return type array_like

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

S

`schnell.correlation`, [7](#)
`schnell.detector`, [3](#)
`schnell.mapping`, [8](#)

D

Detector (*class in schnell.detector*), 3

G

get_antenna() (*schnell.mapping.MapCalculator method*), 9

get_corrmat() (*schnell.correlation.NoiseCorrelationBase method*), 7

get_dsigm2_dnu_t() (*schnell.mapping.MapCalculator method*), 10

get_Fp() (*schnell.detector.Detector method*), 3

get_G_ell() (*schnell.mapping.MapCalculator method*), 8

get_N_ell() (*schnell.mapping.MapCalculator method*), 8

get_Ninv_t() (*schnell.mapping.MapCalculator method*), 9

get_pi_curve() (*schnell.mapping.MapCalculator method*), 10

get_position() (*schnell.detector.Detector method*), 4

get_position() (*schnell.detector.GroundDetector method*), 5

get_position() (*schnell.detector.GroundDetectorTriangle method*), 5

get_position() (*schnell.detector.LISADetector method*), 6

get_transfer() (*schnell.detector.Detector method*), 4

get_u_v() (*schnell.detector.GroundDetector method*), 5

get_u_v() (*schnell.detector.GroundDetectorTriangle method*), 6

get_u_v() (*schnell.detector.LISADetector method*), 6

get_uu_vv() (*schnell.detector.Detector method*), 4

GroundDetector (*class in schnell.detector*), 4

GroundDetectorTriangle (*class in schnell.detector*), 5

L

LISADetector (*class in schnell.detector*), 6

M

MapCalculator (*class in schnell.mapping*), 8

N

NoiseCorrelationBase (*class in schnell.correlation*), 7

NoiseCorrelationConstant (*class in schnell.correlation*), 7

NoiseCorrelationConstantIdentity (*class in schnell.correlation*), 7

NoiseCorrelationConstantR (*class in schnell.correlation*), 7

NoiseCorrelationFromFunctions (*class in schnell.correlation*), 7

NoiseCorrelationLISA (*class in schnell.correlation*), 7

P

psd() (*schnell.detector.Detector method*), 4

psd() (*schnell.detector.LISADetector method*), 6

psd_A() (*schnell.detector.LISADetector method*), 6

psd_X() (*schnell.detector.LISADetector method*), 7

S

schnell.correlation (*module*), 7

schnell.detector (*module*), 3

schnell.mapping (*module*), 8