so$_pysm_modelsDocumentation$

Release 0.2.dev149

A. Zonca

Mar 22, 2019

Contents:

This is the documentation for so_pysm_models.

Part I

Getting started

CHAPTER 1

Installation

Requirements:

- PySM PySM
- healpy

Clone the repository:

```
pip install https://github.com/simonsobs/so_pysm_models/archive/master.zip
```

CHAPTER 2

Development installation

Clone from Github and install:

```
git clone https://github.com/simonsobs/so_pysm_models
cd so_pysm_models
pip install -e .
```

Run unit tests:

```
python setup.py test -V
```

Build docs:

```
python setup.py build_docs -w
```

CHAPTER 3

Example Usage

This repository implements new models for PySM that can be added as additional components.

For example, create and configure a component:

```
from so_pysm_models import GaussianSynchrotron
synchrotron = GaussianSynchrotron(target_nside = 16)
```

Create a PySM sky and add this component:

```
sky = pysm.Sky({})
sky.add_component("gaussian_synch", gaussian_synch)
```

Then get a map at a specific frequency in GHz with standard PySM functionalities:

```
m_synch = sky.gaussian_synch(2.3)
```

see example notebooks:

- Example Gaussian Synchrotron
- Example Gaussian Dust
- Example InterpolatingComponent

Part II

Summary of Models

This page contains high-level documentation about the available models, check the classes docstrings, or the `online documentation`, for the specific arguments.

The input template maps for many models are available at NERSC on the `cmb` project space at:

```
/global/project/projectdirs/cmb/www/so_pysm_models_data
```

they are also published via web at http://portal.nersc.gov/project/cmb/so_pysm_models_data/.

CHAPTER 4

GaussianSynchrotron

This class implements Gaussian simulations for Galactic synchrotron emission. The inputs are a bunch of parameters defining the properties of the synchrotron power spectra, and of synchrotron Spectral Energy Distribution (SED), the output are the stokes IQU maps simulated as Gaussian random fields of the defined spectra. In particular, synchrotron power spectra C_ℓ are assumed to follow a power law as a function of ℓ: $C_\ell^{TT/TE/EE/BB} \propto \ell^\alpha$. Spectra are defined by:

1. The slope α (same for all the spectra)
2. The amplitude of TT and EE spectra at $\ell = 80$,
3. The ratio between B and E-modes

Stokes Q and U maps are generated as random realization of the polarization spectra. For the temperature map the situation is slightly different as we want the total intensity map to be positive everywhere. The Stokes I map is generated in the following way:

if target Nside<=64:

1. The TT power spectrum is $C_\ell \propto \ell^\alpha$ and $C_\ell[0] = 0$
2. A first temparature map T is generated as a gaussian realization of this power spectrum
3. A new map is obtained by adding to T an offset whose value is taken from a reference map
4. If T+offset is positive everywhere than this is the output temperature map
5. Otherwise a cut in the TT power spectrum is applied in the following way: $C_\ell[1 : \ell_{cut}] = C_\ell[\ell_{cut}]$
6. A new T+offset map is generated. The value of ℓ_{cut} is the minimum one for which T+offset is positive everywhere

if target Nside>64:

1. a map at Nside=64 is generated following the procedure above and then filtered to retain only large angular scales (ell<30)

2. a map at the target Nside is generated including only small scales (ell>30) with the same seed as the map at point 1.
3. the two maps are added together
4. In case the coadded map still has negative pixels a small offset is added to make it positive everywhere

The default parameters are optimized for SO-SAT observations. Meaning that the amplitudes of power spectra are normalized in the 10% sky region observed by the instrument. In particular:

1. The amplitude of TT spectrum is taken from PySM-s0 model at 23GHz. TT_amplitude = 20 μK^2 (for D_ℓ at $\ell = 80$) 1. The offset for T map is also taken from PySM-s0 model at 23GHz. Toffset = 72 μK 1. The amplitude of EE spectrum is taken from S-PASS at 2.3GHz extrapolated at 23GHz with a powerlaw with $\beta_s = -3.1$ EE_amplitude = 4.3 math:`mu K^2` (for D_ℓ at $\ell = 80$) 1. ratio between B and E modes from Krachmalnicoff et al. 2018, B_to_E = 0.5 1. spectral tilt from Krachmalnicoff et al 2018, alpha = -1 1. spectral index from Planck IX 2018, beta = -3.1 1. Default value for curvature is zero

CHAPTER 5

GaussianDust

This class implements Gaussian simulations for Galactic thermal dust emission. The inputs are a bunch of parameters defining the properties of dust power spectra, and of dust Spectral Energy Distribution (SED), the output are the stokes IQU maps simulated as Gaussian random fields of the defined spectra. In particular, dust power spectra C_ℓ are assumed to follow a power law as a function of ℓ: $C_\ell^{TT/TE/EE/BB} \propto \ell^\alpha$. Spectra are defined by:

1. The slope α (same for all the spectra)
2. The amplitude of TT and EE spectra at $\ell = 80$,
3. The ratio between B and E-modes
4. The degree of correlation between T and E.

Stokes Q and U maps are generated as random realization of the polarization spectra. For the temperature map the situation is slightly different as we want the total intensity map to be positive everywhere. The Stokes I map is generated in the following way:

if target Nside<=64:

1. The TT power spectrum is $C_\ell \propto \ell^\alpha$ and $C_\ell[0] = 0$
2. A first temparature map T is generated as a gaussian realization of this power spectrum
3. A new map is obtained by adding to T an offset whose value is taken from a reference map
4. If T+offset is positive everywhere than this is the output temperature map
5. Otherwise a cut in the TT power spectrum is applied in the following way: $C_\ell[1 : \ell_{cut}] = C_\ell[\ell_{cut}]$
6. A new T+offset map is generated. The value of ℓ_{cut} is the minimum one for which T+offset is positive everywhere

if target Nside>64:

1. a map at Nside=64 is generated following the procedure above and then filtered to retain only large angular scales (ell<30)

2. a map at the target Nside is generated including only small scales (ell>30) with the same seed as the map at point 1.
3. the two maps are added together
4. In case the coadded map still has negative pixels a small offset is added to make it positive everywhere

Typical values for ℓ_{cut} are between $\ell = 4$ and $\ell = 9$, depending on realization (and also on the Nside of the output map). This implementation removes some power at the very large scales.

The default parameters are optimized for SO-SAT observations. Meaning that the amplitudes of power spectra are normalized in the 10% sky region observed by the instrument. In particular:

1. The amplitude of TT spectrum is taken from PySM-d0 model at 353GHz. TT_amplitude = 350 μK^2 (for D_ℓ at $\ell = 80$)
2. The offset for T map is also taken from PySM-d0 model at 353GHz. Toffset = 18 μK
3. The amplitude of EE spectrum is taken from Planck map at 353GHz, EE_amplitude = 100 math:`mu K^2` (for D_ℓ at $\ell = 80$)
4. ratio between B and E modes from Planck IX 2018, B_to_E = 0.5
5. spectral tilt from Planck IX 2018, alpha = -0.42
6. spectral index and temperature from Planck IX 2018, beta = 1.53, T=19.6 K

CHAPTER 6

COLines

`COLines` is not a standard PySM component because PySM does not allow to distinguish between a case where a component is evaluated for the purpose of integrating over the bandpass or evaluated for separate channels. Therefore this class should be instantiated choosing the desired line and summed to the output of PySM. For example:

```
from so_pysm_models import COLines
co = COLines(target_nside=16, output_units="uK_CMB", line="10")
pysm_map += bandpass_weight * hp.smoothing(co.signal(), fwhm=fwhm)
```

Where `bandpass_weight` is the scalar transmission at the line frequency (which is available at `co.line_frequency`), i.e. if the bandpass is a top-hat between 110 and 120 GHz, the "10" line emission should be multiplied by `0.1`.

This class implements simulations for Galactic CO emission involving the first 3 CO rotational lines, i.e. $J = 1-0, 2-1, 3-2$ whose center frequency is respectively at $\nu_0 = 115.3, 230.5, 345.8$ GHz. The CO emission map templates are the CO Planck maps obtained with `MILCA` component separation algorithm (See `Planck paper`). The CO maps have been released at the nominal resolution (10 and 5 arcminutes). However, to reduce noise contamination from template maps (especially at intermediate and high Galactic latitudes), we convolved them with a 1 deg gaussian beam.

The Stokes I map is computed from the template one as it follows:

if target Nside <= 512:

1. The template map at a `nside=512` is downgraded at the target nside

if target Nside > 512 :

1. The template map at a `nside=2048` is downgraded(eventually upgraded) at the target nside

Q and U maps can be computed from the template CO emission map, I_{CO}, assuming a constant fractional polarization, as:

$$Q = f_{pol} I_{CO} g_d \cos(2\psi)$$
$$U = f_{pol} I_{CO} g_d \sin(2\psi)$$

with g_d and ψ being respectively the depolarization and polarization angle maps estimated from a dust map as :

$$g_d = \frac{\sqrt{Q_{d,353}^2 + U_{d,353}^2}}{f_{pol} I_{d,353}}$$
$$\psi = \frac{1}{2}\arctan\frac{U_{d,353}}{Q_{d,353}}$$

Most of the CO emission is expected to be confined in the Galactic midplane. However, there are still regions at high Galactic latitudes where the CO emission has been purely assessed (by current surveys) and where the Planck signal-to-noise was not enough to detect any emission.

The PySM user can include the eventuality of molecular emission (both unpolarized and polarized) at High Gal. Latitudes by coadding to the emission maps one realization of CO emission simulated with MCMole3D together with the Planck CO map. The polarization is simulated similarly as above.

The `MCMole3D` input parameters are are obtained from best fit with the Planck CO 1-0 map (see Puglisi et al. 2017 and the `documentation`). If `include_high_galactic_latitude_clouds=True`, a mock CO cloud map is simulated with `MCMole3D`, encoding high Galactic latitudes clouds at latitudes above and below than 20 degres. The mock emission map is then coadded to the Planck CO emission map. The polarization is simulated similarly as above.

The installation of `mcmole3d` is not required, HGL clouds can be input to the CO emission by setting `run_mcmole3d=False` (which is the default). However, if one wants to run several mock CO realizations observing high Galactic latitude patches we encourage to run `mcmole3d` by changing `random_seed` in the CO class constructor. The parameter `theta_high_galactic_latitude_deg` set the latitude above which CO emission from high Galactic latitudes can be included and it has an impact **only when** `run_mcmole3d=True`.

The default parameters are set to include CO 1-0 emission and polarization (with 0.1% constant polarization fraction), in particular:

1. `polarization_fraction= 0.001`, on average is the expected level on 10% regions of the sky. However, polarization from CO emission have been detected at larger fluxes in Orion and Taurus complexes (Greaves et al.1999)
2. `theta_high_galactic_latitude_deg = 20`, includes CO emission at $|b| > \theta_{hgl}$ from one realization of mcmole3d maps. Be aware that the larger $theta_{hgl}$, the farther is the Galactic plane and the more unlikely is to find high Galactic latitude clouds.

CHAPTER 7

PrecomputedAlms

This class generates a PySM component based on a set of precomputed $a_{\ell,m}$ coefficients stored in a folder in FITS format. This is mostly targeted at simulations of the Cosmic Microwave Background, the input $a_{\ell,m}$ can be in `K_{RJ}` or `K_{CMB}` as defined in the constructor, the unit conversion is performed assuming the `CMB` black body spectrum. The output unit is specified in the `signal` method, default is `mu K_{RJ}`, as expected by `PySM`. In case the input is in `K_{RJ}`, it is necessary also to specify `input_reference_frequency_GHz`.

The transformation between Spherical Harmonics and pixel domain can be performed either during initialization or in the `signal` method based on `precompute_output_map`.

See the documentation about mapsims about specific simulated datasets.

CHAPTER 8

InterpolatingComponent

Adds a custom emission to the sky simulated by PySM defined as a set of template maps at pre-defined frequencies to be interpolated at the frequencies requested through PySM.

Inputs

A folder of maps named with their frequency in GHz with the flux in any unit supported by PySM (e.g. `Jysr`, `MJsr`, `uK_RJ`, `K_CMB`). They don't need to be equally spaced

For example:

```
ls `cib_precomputed_maps/`
0010.0.fits 0015.0.fits 0018.0.fits
```

Usage

Instantiate `InterpolatingComponent` and point it to the folder, define the unit and the target nside (same used by PySM). It supports all `interpolation_kind` of `scipy.interpolate.interp1d()`, e.g. "nearest", "linear", "quadratic", "cubic":

```
cib = InterpolatingComponent(path="cib_precomputed_maps", input_units="MJysr", target_nside=nside,␣
↪interpolation_kind="linear",
                         has_polarization=False, verbose=True)
```

`Full example notebook`

Part III

High resolution templates

so_pysm_models also provides access to templates with higher resolution and with updated data compared to the models included in PySM.

They can be accessed with the function `get_so_models()` which works similarly to the `models` function available in PySM, and you can mix them, for example:

```
from so_pysm_models import get_so_models
from pysm.nominal import models
from pysm import Sky
sky = Sky({
        "dust" : get_so_models("SO_d0", nside=128),
        "synchrotron" : models("s1", nside=128)
})
```

so_pysm_models retrieves the templates when needed from NERSC via web accessing: http://portal.nersc.gov/project/cmb/so_pysm_models_data/ Downloaded files are stored in the `astropy` cache, generally `astropy/cache` and are accessible using `astropy.utils.data`, e.g. `astropy.utils.data.get_cached_urls()` gives the list of downloaded files. If running at NERSC, the module automatically uses the files accessible locally from the `/project` filesystem.

Low-resolution templates are standard PySM ones at N_{side} 512, often with updated parameters based on Planck results. High-resolution templates are computed from the low-resolution ones, by extrapolating power spectra considering a simple power law model, and by generating small scales as Gaussian realization of these spectra. High-resolution templates therefore have Gaussian small scales (for $\ell > 1000$) modulated with large scale signal for both temperature and polarization.

You can access the high resolution parameters at N_{side} 4096 appending `s` (for small scale) at the end of each model name, for example:

```
from so_pysm_models import get_so_models
from pysm import Sky
sky_highres = Sky({
        "dust" : get_so_models("SO_d0s", nside=4096),
        "synchrotron" : get_so_models("SO_s0s", nside=4096)
})
```

Whatever the N_{side} of the input model and the requested N_{side} in `get_so_models()`, PySM will automatically use `healpy.ud_grade()` to adjust the map resolution.

CHAPTER 9

Details about individual models

Append “s” after a model name to access the N_{side} 4096 template, i.e. `SO_f0s`.

Dust

- **SO_d0**: Thermal dust is modeled as a single-component modified black body, with same templates as in PySM model `d1`. There is no spatial variation of temperature and emissivity in the sky: $T = 19.6$ K and $\beta_d = 1.53$ (values taken from Planck Collaboration IX 2018).

Synchrotron

- **SO_s0**: Templates from PySM model `s1`. Power law spectral energy distribution, with fixed spectral index $\beta_s = -3.1$ (from Planck Collaboration IX 2018).

Free Free

- **SO_f0**: same model as PySM `f1`, no spatial variation of spectral index equal to -2.4.

AME

- **SO_a0**: sum of two spinning dust populations (as in PySM model `a1`) with spatially constant peak frequency. No polarization.

Part IV

Reference/API

CHAPTER 10

so_pysm_models Package

10.1 Functions

`get_so_models`(key, nside[, pixel_indices, . . .])	
`test`(**kwargs)	Run the tests for the package.

10.1.1 get_so_models

`so_pysm_models.`**`get_so_models`**(*key*, *nside*, *pixel_indices=None*, *mpi_comm=None*)

10.1.2 test

`so_pysm_models.`**`test`**(***kwargs*)

Run the tests for the package.

This method builds arguments for and then calls `pytest.main`.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. Accepts comma separated string to specify multiple packages. If nothing is specified all default tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int or 'auto', optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is `'auto'`, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

pastebin
[('failed', 'all', None), optional] Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @pytest.mark.remote_data. This can be set to run no tests with remote data (`none`), only ones that use data from http://data.astropy.org (`astropy`), or all tests that use remote data (`any`). The default is `none`.

repeat
[`int`, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[`bool`, optional] When `True`, skips running the doctests in the .rst files.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

10.2 Classes

`COLines`(target_nside, output_units[, ...])	Class defining attributes for CO line emission.
`GaussianDust`(target_nside[, ...])	Gaussian dust model
`GaussianSynchrotron`(target_nside[, ...])	Gaussian synchrotron model
`InterpolatingComponent`(path, input_units, ...)	PySM component interpolating between precomputed maps
`PrecomputedAlms`(filename[, input_units, ...])	Generic component based on Precomputed Alms
`UnsupportedPythonError`	

10.2.1 COLines

class so_pysm_models.**COLines**(*target_nside*, *output_units*, *has_polarization=True*, *line='10'*, *include_high_galactic_latitude_clouds=False*, *polarization_fraction=0.001*, *theta_high_galactic_latitude_deg=20.0*, *random_seed=1234567*, *verbose=False*, *run_mcmole3d=False*, *pixel_indices=None*, *mpi_comm=None*)

Bases: `object`

Class defining attributes for CO line emission. CO templates are extracted from Type 1 CO Planck maps. See further details in https://www.aanda.org/articles/aa/abs/2014/11/aa21553-13/aa21553-13.html

Parameters

target_nside
[int] HEALPix NSIDE of the output maps

output_units
[str] unit string as defined by `pysm.convert_units`, e.g. uK_RJ, K_CMB

has_polarization
[bool] whether or not to simulate also polarization maps

line
[string] CO rotational transitions. Accepted values : 10, 21, 32

polarization_fraction: float
polarisation fraction for polarised CO emission.

include_high_galactic_latitude_clouds: bool
If True it includes a simulation from MCMole3D to include high Galactic Latitude clouds. (See more details at http://giuspugl.github.io/mcmole/index.html)

run_mcmole3d: bool
If True it simulates HGL cluds by running MCMole3D, otherwise it coadds a map of HGL emission.

random_seed: int
set random seed for mcmole3d simulations.

theta_high_galactic_latitude_deg
[float] Angle in degree to identify High Galactic Latitude clouds (i.e. clouds whose latitude b is `|b|> theta_high_galactic_latitude_deg`).

pixel_indices
[ndarray of ints] Outputs partial maps given HEALPix pixel indices in RING ordering

mpi_comm
[mpi4py communicator] Read inputs across a MPI communicator, see pysm.read_map

Methods Summary

`read_map`(fname[, field])	
`signal`()	Simulate CO signal
`simulate_high_galactic_latitude_CO`()	Coadd High Galactic Latitude CO emission, simulated with MCMole3D.

Continued on next page

Table 3 – continued from previous page

`simulate_polarized_emission`(I_map)	Add polarized emission by means of: * an overall constant polarization fraction, * a depolarization map to mimick the line of sigth depolarization effect at low Galactic latitudes * a polarization angle map coming from a dust template (we exploit the observed correlation between polarized dust and molecular emission in star forming regions).

Methods Documentation

`read_map`(*fname*, *field=None*)

`signal`()

Simulate CO signal

`simulate_high_galactic_latitude_CO`()

Coadd High Galactic Latitude CO emission, simulated with MCMole3D.

`simulate_polarized_emission`(*I_map*)

Add polarized emission by means of: * an overall constant polarization fraction, * a depolarization map to mimick the line of sigth depolarization effect at low Galactic latitudes * a polarization angle map coming from a dust template (we exploit the observed correlation between polarized dust and molecular emission in star forming regions).

10.2.2 GaussianDust

class `so_pysm_models.`**`GaussianDust`**(*target_nside*, *has_polarization=True*, *pixel_indices=None*, *TT_amplitude=350.0*, *Toffset=18.0*, *EE_amplitude=100.0*, *rTE=0.35*, *EtoB=0.5*, *alpha=-0.42*, *beta=1.53*, *temp=19.6*, *nu_0=353*, *seed=None*)

Bases: `object`

Gaussian dust model

See more details at https://so-pysm-models.readthedocs.io/en/latest/so_pysm_models/models.html

Parameters

target_nside
[int] HEALPix NSIDE of the output maps

has_polarization
[bool] whether or not to simulate also polarization maps Default: True

pixel_indices
[ndarray of ints] Outputs partial maps given HEALPix pixel indices in RING ordering

TT_amplitude
[float] amplitude of synchrotron TT power spectrum (D_ell) at at the reference frequency and ell=80, in muK^2 and thermodinamic units. Default: 350. from the amplitude of PySM-d0 dust model at 353GHz in the region covered by SO-SAT.

Toffset
[float] offset to be applied to the temperature map in muK in RJ units. Default: 18 from the mean value of the T PySM-s0 synch map at 23GHz in the region covered by SO-SAT

EE_amplitude
[float] Amplitude of EE modes D_ell at reference frequency at ell=80 Default: 100. from the amplitude of HFI-353 E-modes spectrum in the region covered by SO-SAT

EtoB: float
ratio between E and B-mode amplitude for dust. Default: 0.5 from Planck 2018 IX

alpha
[same as alpha_sync for dust.] Default: -0.42 from Planck 2018 IX

beta
[float] dust spectral index. Default: 1.53 from Planck 2018 IX

temp
[float] dust temperature. Default: 19.6 from Planck 2018 IX

nu0
[float] dust reference frequency in GHz. Default: 353

seed
[int] seed for random realization of map Default: None

Methods Summary

`signal`(nu, **kwargs)	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

`signal`(*nu*, ***kwargs*)
Return map in uK_RJ at given frequency or array of frequencies

10.2.3 GaussianSynchrotron

class so_pysm_models.**GaussianSynchrotron**(*target_nside*, *has_polarization=True*, *pixel_indices=None*, *TT_amplitude=20.0*, *Toffset=72.0*, *EE_amplitude=4.3*, *rTE=0.35*, *EtoB=0.5*, *alpha=-1.0*, *beta=-3.1*, *curv=0.0*, *nu_0=23.0*, *seed=None*)

Bases: `object`

Gaussian synchrotron model

See more details at https://so-pysm-models.readthedocs.io/en/latest/so_pysm_models/models.html

Parameters

target_nside
[int] HEALPix NSIDE of the output maps

has_polarization
[bool] whether or not to simulate also polarization maps Default: True

pixel_indices
[ndarray of ints] Outputa partial maps given HEALPix pixel indices in RING ordering

TT_amplitude
[float] amplitude of synchrotron TT power spectrum (D_ell) at at the reference frequency

and ell=80, in muK^2 and thermodinamic units. Default: 20 from the amplitude of PySM-s0 synchrotron model at 23GHz in the region covered by SO-SAT.

Toffset
[float] offset to be applied to the temperature map in muK. Default: 72 from the mean value of the T PySM-s0 synch map at 23GHz in the region covered by SO-SAT

EE_amplitude
[float] same as TT_amplitude but for EE power spectrum. Default: 4.3 from the amplitude of S-PASS E-modes power spectrum at 2.3GHz in the region covered by SO-SAT, rescaled at 23GHz with a powerlaw with beta_s = -3.1

rTE
[float] TE correlation factor defined as: rTE = clTE/sqrt(clTT*clEE) Default: 0.35 from Planck IX 2018

EtoB
[float] ratio between E and B-mode amplitude. Default: 0.5 from Krachmalnicoff et al. 2018

alpha
[spectral tilt of the synchrotron power spectrum (D_ell).] Default: -1.0 from Krachmalnicoff et al. 2018

beta
[synchrotron spectral index.] Default: -3.1 from Planck 2018 IX

curv
[synchrotron curvature index.] Default: 0.

nu_0
[synchrotron reference frequency in GHz.] Default: 23

seed
[int] seed for random realization of map Default: None

Methods Summary

`signal`(nu, **kwargs)	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

`signal`(*nu*, ***kwargs*)
Return map in uK_RJ at given frequency or array of frequencies

10.2.4 InterpolatingComponent

class so_pysm_models.**InterpolatingComponent**(*path*, *input_units*, *target_nside*, *interpolation_kind='linear'*, *has_polarization=True*, *pixel_indices=None*, *mpi_comm=None*, *verbose=False*)

Bases: `object`

PySM component interpolating between precomputed maps

See more details at https://so-pysm-models.readthedocs.io/en/latest/so_pysm_models/models.html

Parameters

path
[str] Path should contain maps named as the frequency in GHz e.g. 20.fits or 20.5.fits or 00100.fits

input_units
[str] Any unit available in PySM (see `pysm.convert_units` e.g. `Jysr`, `MJsr`, `uK_RJ`, `K_CMB`).

target_nside
[int] HEALPix NSIDE of the output maps

has_polarization
[bool] whether or not to simulate also polarization maps

pixel_indices
[ndarray of ints] Outputs partial maps given HEALPix pixel indices in RING ordering

mpi_comm
[mpi4py communicator] See the documentation of pysm.read_map

verbose
[bool] Control amount of output

Methods Summary

`get_filenames`(path)	
`read_map`(freq)	
`signal`(nu, **kwargs)	Return map at given frequency or array of frequencies

Methods Documentation

`get_filenames`(*path*)

`read_map`(*freq*)

`signal`(*nu*, ***kwargs*)
Return map at given frequency or array of frequencies

10.2.5 PrecomputedAlms

`class so_pysm_models.`**`PrecomputedAlms`**(*filename*, *input_units='uK_CMB'*, *input_reference_frequency_GHz=None*, *target_nside=None*, *target_shape=None*, *target_wcs=None*, *precompute_output_map=True*, *has_polarization=True*, *pixel_indices=None*)

Bases: `object`

Generic component based on Precomputed Alms

Load a set of Alms from a FITS file and generate maps at the requested resolution and frequency assuming the CMB black body spectrum. A single set of Alms is used for all frequencies requested by PySM, consider that

PySM expects the output of components to be in uK_RJ.

See more details at https://so-pysm-models.readthedocs.io/en/latest/so_pysm_models/models.html

Parameters

filename
[string] Path to the input Alms in FITS format

input_units
[string] Input unit strings as defined by pysm.convert_units, e.g. K_CMB, uK_RJ, MJysr

input_reference_frequency_GHz
[float] If input units are K_RJ or Jysr, the reference frequency

target_nside
[int] HEALPix NSIDE of the output maps

precompute_output_map
[bool] If True (default), Alms are transformed into a map in the constructor, if False, the object only stores the Alms and generate the map at each call of the signal method, this is useful to generate maps convolved with different beams

has_polarization
[bool] whether or not to simulate also polarization maps Default: True

pixel_indices
[ndarray of ints] Output a partial maps given HEALPix pixel indices in RING ordering

Methods Summary

`compute_output_map`(alm)	
`signal`([nu, fwhm_arcmin, output_units])	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

`compute_output_map`(*alm*)

`signal`(*nu=[148.0], fwhm_arcmin=None, output_units='uK_RJ', **kwargs*)
Return map in uK_RJ at given frequency or array of frequencies

If nothing is specified for nu, we default to providing an unmodulated map at 148 GHz. The value 148 Ghz does not matter if the output is in uK_RJ.

Parameters

nu
[list or ndarray] Frequency or frequencies in GHz at which compute the signal

fwhm_arcmin
[float (optional)] Smooth the input alms before computing the signal, this can only be used if the class was initialized with `precompute_output_map` to False.

output_units
[str] Output units, as defined in `pysm.convert_units`, by default this is "uK_RJ" as expected by PySM.

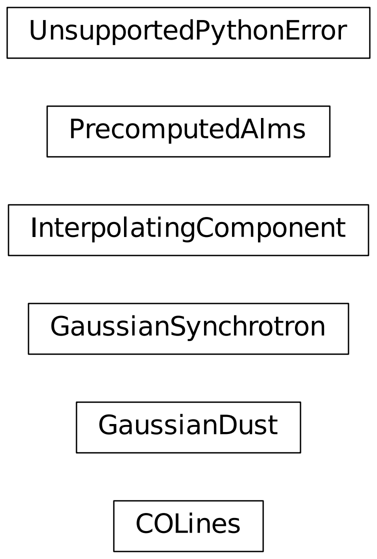
Returns

output_maps
[ndarray] Output maps array with the shape (num_freqs, 1 or 3 (I or IQU), npix)

10.2.6 UnsupportedPythonError

exception so_pysm_models.**UnsupportedPythonError**

10.3 Class Inheritance Diagram



Python Module Index

Index