SO$_{p}ysm_{m}odelsDocumentation$

Release 2.0.dev305

A. Zonca

Sep 23, 2019

CONTENTS:

I Getting started 3

1 Installation 5

2 Development installation 7

3 Example Usage 9

II Summary of Models 11

4 GaussianSynchrotron 15

5 GaussianDust 17

6 COLines 19

7 PrecomputedAlms 21

8 InterpolatingComponent 23

9 WebSky 25

III High resolution templates 27

10 Details about individual models 31

IV Reference/API 33

11 so_pysm_models Package 35

Python Module Index 45

Index 47

This is the documentation for so_pysm_models.

Part I

Getting started

CHAPTER

ONE

INSTALLATION

Requirements:

- PySM 3 PySM
- healpy

Install with `pip` from Github:

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

CHAPTER

TWO

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

THREE

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(nside = 16)
```

Create a PySM sky and add this component:

```
sky = pysm.Sky(nside=64, component_objects=[synchrotron])
```

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

```
import astropy.units as u
m_synch = sky.get_emission(2.3 * u.GHz)
```

see example notebooks:

- Example Gaussian Synchrotron
- Example Gaussian Dust
- Example InterpolatingComponent
- Example Websky Extragalactic

Part II

Summary of Models

This page contains high-level documentation about the available models, check the classes doc strings, 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

FOUR

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 $N_{side} <= 64$**:**

1. The TT power spectrum is $C_\ell \propto \ell^\alpha$ and $C_\ell[0] = 0$
2. A first temperature 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 $N_{side} > 64$**:**

1. a map at $N_{side} = 64$ is generated following the procedure above and then filtered to retain only large angular scales (ell<30)
2. a map at the target N_{side} 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 co-added 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 power-law 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

FIVE

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 $N_{side} <= 64$**:**

1. The TT power spectrum is $C_\ell \propto \ell^\alpha$ and $C_\ell[0] = 0$
2. A first temperature 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 $N_{side} > 64$**:**

1. a map at $N_{side} = 64$ is generated following the procedure above and then filtered to retain only large angular scales (ell<30)
2. a map at the target N_{side} 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 co-added 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 N_{side} 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

SIX

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(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 N_{side} <= 512:

1. The template map at a `nside=512` is downgraded at the target N_{side}

if target N_{side} > 512 :

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

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 co-adding 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 degrees. The mock emission map is then co-added 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

SEVEN

PRECOMPUTEDALMS

This class generates a PySM component based on a set of pre-computed $a_{\ell,m}$ 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

EIGHT

INTERPOLATINGCOMPONENT

Moved to PySM 3, FIXME port this documentation over

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 N_{side} (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

CHAPTER

NINE

WEBSKY

The Websky suite of simulated extragalactic component maps, determined from large scale structure light cone realizations and based on Lagrangian perturbation theory, Peak Patch Lagrangian halo finding, and modeling of SZ and CIB effects, can be read into PySM as precomputed external fits files using InterpolatingComponent. More information on the Peak Patch halo finding method can be found in Stein, Alvarez, and Bond (2018), and selected maps and halo catalogs are available from the Websky website. Some additional Websky-specific information and tools are available at the SO Websky model repository.

Specific maps generated for so_pysm_models are described below and located on NERSC at `/project/projectdirs/sobs/v4_sims/mbs/websky/0.3`.

Cosmic Infrared Background

The Planck (2013) CIB halo model is used, along with a halo occupation distribution. More details can be found here.

The current version of the maps are of intensity in units of MJy/Sr with filename convention `cib_nu[FREQ].fits` e.g. `cib_nu0027.fits` is the map of CIB intensity at 27 GHz and will be used by `InterpolatingComponent()` at that frequency, and can be found on NERSC at `/project/projectdirs/sobs/v4_sims/mbs/websky/0.3`. There are 18 fits files at $N_{side} = 4096$ at frequencies [27, 39, 93, 145, 225, 280] +/- 1 GHz, in addition to those corresponding to the Planck HFI channel centers, [100, 143, 217, 353, 545, 857] GHz, for a total of 24 files. These intensities were selected because in order to be able to interpolate accurately at the 6 frequencies of interest with as few maps as possible. More frequencies will be made available after a full set of map based simulations at SO bands that include correlated lensing, CIB, and SZ effects has been generated.

Thermal SZ Effect

Provided is a map of the Compton-y parameter and is based on Battaglia et al. (2012) pressure profiles, and can be found at `/project/projectdirs/sobs/v4_sims/mbs/websky/0.3/tsz.fits`.

Kinetic SZ Effect

Provided is a map of the temperature fluctuation due to line of sight peculiar velocities of electrons along the line of sight. Electrons are assumed to follow a Navarro Frenk and White (NFW) profile interior to halos and second order Lagrangian Perturbation Theory (LPT) outside. The ksz map can be found at `/project/projectdirs/sobs/v4_sims/mbs/websky/0.3/ksz.fits`.

Lensing Convergence

A lensing convergence map is generated from the simulated matter distribution along the line of sight, assumed to follow an NFW profile interior to halos and second order LPT outside.

Primary and lensed CMB The convergence map is used to lens a Gaussian realization of the unlensed primary CMB, which is then read into PySM as the primary lensed CMB through the PrecomputedAlms class. The primary CMB is obtained from parameters that match the Websky simulation, namely A_s = 2.022e-9, tau = 0.055, and all other parameters set to the websky values above. The CAR maps (where the lens remapping is done) have 1 arcminute resolution.

The theoretical power spectra for the unlensed and lensed CMB are available here https://github.com/ajvanengelen/webskylensing/tree/master/data. Each is a numpy array of shape `(3, 3, N_l)`, giving the theory power spectrum C_l's in the order ((TT, TE, TB), (ET, EE, EB), (BT, BE, BB)) in units of uK_CMB^2. They are obtained from the `get_cmb_powerspecta.websky_cmb_spectra` routine in that repository, which serves as a wrapper to CAMB.

Part III

High resolution templates

Starting from version 2.0, all input templates have been rotated to Equatorial, therefore by default the output is in Equatorial coordinates.

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, for example:

```
from so_pysm_models import get_so_models
from pysm.nominal import models
from pysm import Sky
sky = Sky(component_objects=[get_so_models("SO_d0s", nside=4096)])
```

Consider that the so_pysm_models models are by default in **Equatorial coordinates**, therefore they should not be mixed in the same run with standard PySM components which instead are in **Galactic coordinates**. If you need PySM to simultaneously handle inputs in different reference frames, please open an issue in the PySM repository.

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(component_objects=[get_so_models("SO_d0s", 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

TEN

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).
- **SO_d1**: Thermal dust is modeled as a single-component modified black body, with same templates as in PySM model `d1`. Both spectral index and dust temperature are spatially varying up to the degree scale.

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).
- **SO_s1**: Templates from PySM model `s1`. Power law spectral energy distribution, with spatially varying spectral index up to the degree scale.

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.
- **SO_a1**: sum of two spinning dust populations (as in PySM model `a1`). First one with spatially constant peak frequency, the other with spatially variable peak frequency up to the degree scale. Polarized maps simulated with thermal dust angles and nominal AME intensity, scaled globally by 1% polarization fraction.

Part IV

Reference/API

CHAPTER

ELEVEN

SO_PYSM_MODELS PACKAGE

11.1 Functions

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

11.1.1 get_so_models

so_pysm_models.**get_so_models**(*key*, *nside*, *map_dist=None*, *coord='C'*)

11.1.2 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`.

11.2 Classes

COLines(target_nside, output_units[, ...])	Class defining attributes for CO line emission.
GaussianDust(nside[, has_polarization, ...])	Gaussian dust model
GaussianSynchrotron(nside[, ...])	Gaussian synchrotron model
PrecomputedAlms(filename[, input_units, ...])	Generic component based on Precomputed Alms
UnsupportedPythonError	
WebSkyCIB([websky_version, input_units, ...])	PySM component interpolating between precomputed maps
WebSkyCMB(websky_version, nside[, ...])	
WebSkyCMBMap(websky_version, nside[, ...])	
WebSkySZ([version, sz_type, nside, ...])	

11.2.1 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*, *map_dist=None*, *coord='C'*)

Bases: `pysm.models.template.Model`

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`).

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

Methods Summary

`signal`(self)	Simulate CO signal
`simulate_high_galactic_latitude_CO`(self)	Coadd High Galactic Latitude CO emission, simulated with MCMole3D.

Continued on next page

Table 3 – continued from previous page

simulate_polarized_emission(self, 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

signal(*self*)
Simulate CO signal

simulate_high_galactic_latitude_CO(*self*)
Coadd High Galactic Latitude CO emission, simulated with MCMole3D.

simulate_polarized_emission(*self*, *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).

11.2.2 GaussianDust

class so_pysm_models.**GaussianDust**(*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*, *map_dist=None*)
Bases: pysm.models.template.Model

Gaussian dust model

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

Parameters

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

get_emission(self, freqs[, weights])	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

get_emission(*self*, *freqs: Unit("GHz")*, *weights=None*) -> *Unit("uK_RJ")*
Return map in uK_RJ at given frequency or array of frequencies

11.2.3 GaussianSynchrotron

class so_pysm_models.**GaussianSynchrotron**(*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*, *map_dist=None*)

Bases: pysm.models.template.Model

Gaussian synchrotron model

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

Parameters

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

get_emission(self, freqs[, weights])	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

get_emission(*self*, *freqs: Unit("GHz")*, *weights=None) -> Unit("uK_RJ"*)
Return map in uK_RJ at given frequency or array of frequencies

11.2.4 PrecomputedAlms

class so_pysm_models.**PrecomputedAlms**(*filename*, *input_units='uK_CMB'*, *input_reference_frequency=None*, *nside=None*, *precompute_output_map=True*, *has_polarization=True*, *map_dist=None*)
Bases: pysm.models.template.Model

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: float
If input units are K_RJ or Jysr, the reference frequency

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

Methods Summary

compute_output_map(self, alm)	
get_emission(self, freqs, fwhm, None] = None)	Return map in uK_RJ at given frequency or array of frequencies

Methods Documentation

compute_output_map(*self*, *alm*)

get_emission(*self*, *freqs: Unit("GHz")*, *fwhm: [Unit("arcmin")*, *None] = None*, *weights=None) -> Unit("uK_RJ")*
Return map in uK_RJ at given frequency or array of frequencies

Parameters

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

fwhm
[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)

11.2.5 UnsupportedPythonError

exception so_pysm_models.**UnsupportedPythonError**

11.2.6 WebSkyCIB

class so_pysm_models.**WebSkyCIB**(*websky_version='0.3'*, *input_units='MJy / sr'*, *nside=4096*, *interpolation_kind='linear'*, *map_dist=None*, *verbose=False*, *local_folder=None*, *coord='C'*)

Bases: pysm.models.interpolating.InterpolatingComponent

PySM component interpolating between precomputed maps

Methods Summary

get_filenames(self, path)	Get filenames for a websky version For a standard interpolating component, we list files in folder, here we need to know the names in advance so that we can only download the required maps
read_map_by_frequency(self, freq)	

Methods Documentation

get_filenames(*self*, *path*)

Get filenames for a websky version For a standard interpolating component, we list files in folder, here we need to know the names in advance so that we can only download the required maps

read_map_by_frequency(*self*, *freq*)

11.2.7 WebSkyCMB

class so_pysm_models.**WebSkyCMB**(*websky_version*, *nside*, *precompute_output_map=False*, *seed=1*, *lensed=True*, *map_dist=None*, *coord='C'*)

Bases: so_pysm_models.PrecomputedAlms

11.2.8 WebSkyCMBMap

class so_pysm_models.**WebSkyCMBMap**(*websky_version*, *nside*, *precompute_output_map=False*, *seed=1*, *lensed=True*, *include_solar_dipole=False*, *map_dist=None*, *coord='C'*)

Bases: pysm.models.cmb.CMBMap

11.2.9 WebSkySZ

class so_pysm_models.**WebSkySZ**(*version='0.3'*, *sz_type='kinetic'*, *nside=4096*, *map_dist=None*, *verbose=False*, *coord='C'*)

Bases: pysm.models.template.Model

Methods Summary

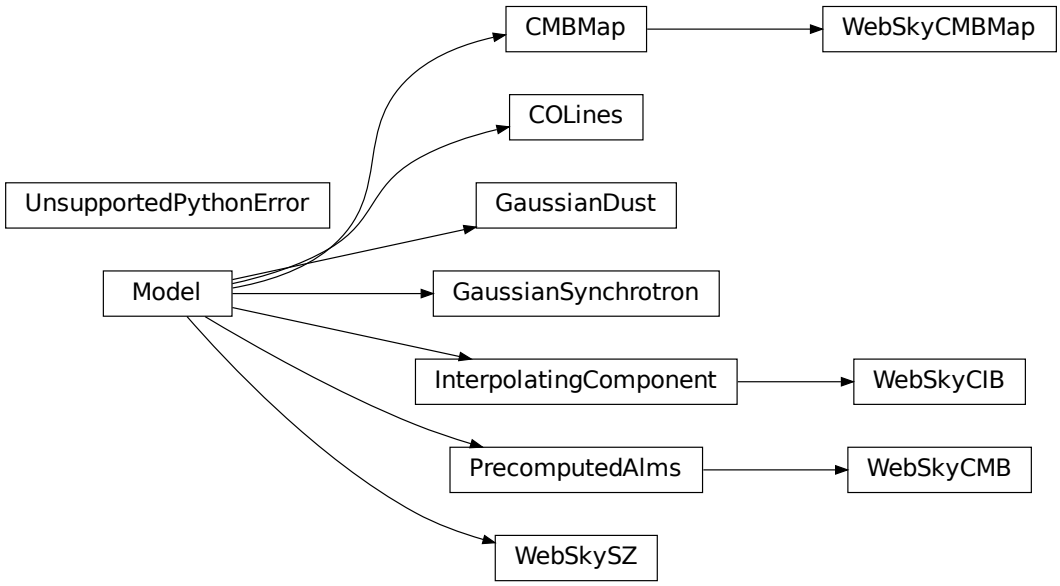
get_emission(self, freqs[, weights])	
get_filename(self)	Get SZ filenames for a websky version

Methods Documentation

get_emission(*self*, *freqs: Unit("GHz")*, *weights=None) -> Unit("uK_RJ")*

get_filename(*self*)
Get SZ filenames for a websky version

11.3 Class Inheritance Diagram



PYTHON MODULE INDEX

S

so_pysm_models, 35

INDEX

C

COLines (*class in so_pysm_models*), 37
compute_output_map() (*so_pysm_models.PrecomputedAlms method*), 41

G

GaussianDust (*class in so_pysm_models*), 38
GaussianSynchrotron (*class in so_pysm_models*), 39
get_emission() (*so_pysm_models.GaussianDust method*), 39
get_emission() (*so_pysm_models.GaussianSynchrotron method*), 40
get_emission() (*so_pysm_models.PrecomputedAlms method*), 41
get_emission() (*so_pysm_models.WebSkySZ method*), 43
get_filename() (*so_pysm_models.WebSkySZ method*), 43
get_filenames() (*so_pysm_models.WebSkyCIB method*), 42
get_so_models() (*in module so_pysm_models*), 35

P

PrecomputedAlms (*class in so_pysm_models*), 40

R

read_map_by_frequency() (*so_pysm_models.WebSkyCIB method*), 42

S

signal() (*so_pysm_models.COLines method*), 38
simulate_high_galactic_latitude_CO() (*so_pysm_models.COLines method*), 38
simulate_polarized_emission() (*so_pysm_models.COLines method*), 38
so_pysm_models (*module*), 35

T

test() (*in module so_pysm_models*), 35

U

UnsupportedPythonError, 42

W

WebSkyCIB (*class in so_pysm_models*), 42
WebSkyCMB (*class in so_pysm_models*), 42
WebSkyCMBMap (*class in so_pysm_models*), 42
WebSkySZ (*class in so_pysm_models*), 42