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

***Release 0.1.dev88***

**A. Zonca**

**Feb 21, 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 PrecomputedAlms 19

III Reference/API 21

7 so_pysm_models Package 23

Python Module Index 31

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

# 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.

# 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 $C_\ell$ are assumed to follow a power law as a function of $\ell$: $C_\ell^{TT/TE/EE/BB} \propto \ell^\alpha$. Spectra are defined by:

1. The slope $\alpha$ (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 $\ell_{cut}$ 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 $\mu K^2$ (for $D_\ell$ at $\ell = 80$) 1. The offset for T map is also taken from PySM-s0 model at 23GHz. Toffset = 72 $\mu K$ 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 $D_\ell$ 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 $C_\ell$ are assumed to follow a power law as a function of $\ell$: $C_\ell^{TT/TE/EE/BB} \propto \ell^\alpha$. Spectra are defined by:

1. The slope $\alpha$ (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 $\ell_{cut}$ 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 $\ell_{cut}$ 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 $\mu K^2$ (for $D_\ell$ at $\ell = 80$)
2. The offset for T map is also taken from PySM-d0 model at 353GHz. Toffset = 18 $\mu K$
3. The amplitude of EE spectrum is taken from Planck map at 353GHz, EE_amplitude = 100 math:`mu K^2` (for $D_\ell$ 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

## 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.

# Part III

# Reference/API

CHAPTER 7

# so_pysm_models Package

## 7.1 Functions

| | |
|---|---|
| test(**kwargs) | Run the tests for the package. |

### 7.1.1 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`.

## 7.2 Classes

| | |
|---|---|
| `GaussianDust`(target_nside[, ...]) | Gaussian dust model |
| `GaussianSynchrotron`(target_nside[, ...]) | Gaussian synchrotron model |
| `PrecomputedAlms`(filename[, input_units, ...]) | Generic component based on Precomputed Alms |
| `UnsupportedPythonError` | |

### 7.2.1 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

### 7.2.2 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

### 7.2.3 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.

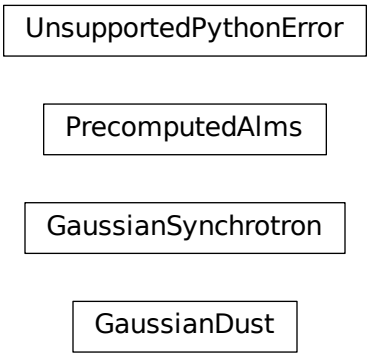
**Returns**

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

### 7.2.4 UnsupportedPythonError

**exception** so_pysm_models.**UnsupportedPythonError**

## 7.3 Class Inheritance Diagram



# Python Module Index

## S

so_pysm_models, 23

# Index


## C

compute_output_map() (*so_pysm_models.PrecomputedAlms method*), 28

## G

GaussianDust (*class in so_pysm_models*), 24
GaussianSynchrotron (*class in so_pysm_models*), 25

## P

PrecomputedAlms (*class in so_pysm_models*), 27

## S

signal() (*so_pysm_models.GaussianDust method*), 25
signal() (*so_pysm_models.GaussianSynchrotron method*), 27
signal() (*so_pysm_models.PrecomputedAlms method*), 28
so_pysm_models (*module*), 23

## T

test() (*in module so_pysm_models*), 23

## U

UnsupportedPythonError, 28