PySM Documentation

Release 2.0

Ben Thorne, David Alonso, Jo Dunkley, Sigurd Naess

Nov 07, 2019

Contents:

CHAPTER 1

Introduction

This code generates full-sky simulations of Galactic foregrounds in intensity and polarization relevant for CMB experiments. The components simulated are: thermal dust, synchrotron, AME, free-free, and CMB at a given Nside, with an option to integrate over a top hat bandpass, to add white instrument noise, and to smooth with a given beam.

There is scope for a few options for the model for each component, attempting to be consistent with current data. The current v-1.0 version has typically two-three options for each component.

Currently much of the available data is limited in resolution at degree-scale. We therefore make efforts to provide reasonable small-scale simulations to extend the data to higher multipoles. The details of the procedures developed can be found in the accompanying paper.

This code is based on the large-scale Galactic part of Planck Sky Model (Delabrouille 2012) code and uses some of its inputs.

1.1 Dependencies

PySM is written in Python and uses the healpy, numpy, scipy, and astropy packages. It is known to work with:

- python 2.7.6
- healpy 1.10.3
- numpy 1.12.1
- scipy 0.19.0

1.2 Installation

Clone the GitHub repository and run:

```
[sudo] python setup.py install [--user]
```

After this you can run the provided unit tests from the same directory:

```
nosetests
```

Then you may import PySM in the standard way in a Python environment:

```
import pysm
```

CHAPTER 2

Summary of Models

2.1 Dust

- **d1**: Thermal dust is modelled as a single-component modified black body (mbb). We use dust templates for emission at 545 GHz in intensity and 353 GHz in polarisation from the Planck-2015 analysis, and scale these to different frequencies with a mbb spectrum using the spatially varying temperature and spectral index obtained from the Planck data using the Commander code (Planck Collaboration 2015, arXiv:1502.01588). Note that it therefore assumes the same spectral index for polarization as for intensity. The input intensity template at 545 GHz is simply the available 2048 product degraded to nside 512. The polarization templates have been smoothed with a Gaussian kernel of FWHM 2.6 degrees, and had small scales added via the procedure described in the accompanying paper.
- **d2** (**d3**): emissivity that varies spatially on degree scales, drawn from a Gaussian with beta=1.59 pm 0.2 (0.3). A Gaussian variation is not physically motivated, but amount of variation consistent with Planck.
- **d4**: a generalization of model 1 to multiple dust populations. It has been found that a two component model is still a good fit to the Planck data. This option uses the two component model from Finkbeiner, D. P., Davis, M., & Schlegel, D. J. 1999, Astrophysical Journal, 524, 867.
- **d5**: implementation of the dust model described in Hensley and Draine 2017.
- **d6**: implementation of the frequency decorrelation of dust, modelling the impact of averaging over spatially varying dust spectral indices both unresolved and along the line of sight. We take an analytic frequency covariance (Vansyngel 2016 arXiv:1611.02577) to calculate the resulting frequency dependence. The user specifies a single parameter, the correlation length. The smaller the correlation length, the larger the decorrelation. This parameter is constant across the sky.

2.2 Synchrotron

- **s1**: A power law scaling is used for the synchrotron emission, with a spatially varying spectral index. The emission templates are the Haslam 408 MHz, 57' resolution data reprocessed by Remazeilles et al 2015 MNRAS 451, 4311, and the WMAP 9-year 23 GHz Q/U maps (Bennett, C.L., et.al., 2014, ApJS, 208, 20B). The polarization maps have been smoothed with a Gaussian kernel of FWHM 5 degrees and had small scales added. The

intensity template has had small scales added straight to the template. The details of the small scale procedure is outlined in the accompanying paper. The spectral index map was derived using a combination of the Haslam 408 MHz data and WMAP 23 GHz 7-year data (Miville-Deschenes, M.-A. et al., 2008, A&A, 490, 1093). The same scaling is used for intensity and polarization. This is the same prescription as used in the Planck Sky Model's v1.7.8 'power law' option (Delabrouille et al. A&A 553, A96, 2013), but with the Haslam map updated to the Remazeilles version. A 'curved power law' model is also supported with a single isotropic curvature index. The amplitude of this curvature is taken from Kogut, A. 2012, ApJ, 753, 110.

- **s2**: synchrotron index steepens off the Galactic plane, from -3.0 in the plane to -3.3 off the plane. Consistent with WMAP.
- **s3**: a power law with a curved index. The model uses the same index map as the nominal model, plus a curvature term. We use the best-fit curvature amplitude of -0.052 found in Kogut, A. 2012, ApJ, 753, 110, pivoted at 23 GHz.

2.3 AME

- **a1**: We model the AME as a sum of two spinning dust populations based on the Commander code (Planck Collaboration 2015, arXiv:1502.01588). A component is defined by a degree-scale emission template at a reference frequency and a peak frequency of the emission law. Both populations have a spatially varying emission template, one population has a spatially varying peak frequency, and the other population has a spatially constant peak frequency. The emission law is generated using the SpDust2 code (Ali-Haimoud 2008). The nominal model is unpolarized. We add small scales to the emission maps, the method is outlined in the accompanying paper.
- **a2**: AME has 2% polarization fraction. Polarized maps simulated with thermal dust angles and nominal AME intensity scaled globally by polarization fraction. Within WMAP/Planck bounds.

2.4 Free-free

- **f1**: We model the free-free emission using the analytic model assumed in the Commander fit to the Planck 2015 data (Draine 2011 'Physics of the Interstellar and Intergalactic Medium') to produce a degree-scale map of free-free emission at 30 GHz. We add small scales to this using a procedure outlined in the accompanying paper. This map is then scaled in frequency by applying a spatially constant power law index of -2.14.

2.5 CMB

- **c1**: A lensed CMB realisation is computed using Taylens, a code to compute a lensed CMB realisation using nearest-neighbour Taylor interpolation (taylens; Naess, S. K. and Louis, T. JCAP 09 001, 2013, astro-ph/1307.0719). This code takes, as an input, a set of unlensed Cl's generated using CAMB. The params.ini is in the Ancillary directory. There is a pre-computed CMB map provided at Nside 512.

CHAPTER 3

Ho to use PySM

3.1 Sky object

The central object of PySM is the `pysm.pysm.Sky` object. This is initialised using a dictionary in which we specify the required models:

```
import pysm
from pysm.nominal import models

sky_config = {
'dust' : [dust_pop_1, dust_pop_2, ...],
'synchrotron' : [synch_pop_1, synch_pop_2, ...],
'ame' : [ame_pop_1, ame_pop_2, ...],
'freefree' : [ff_pop_1, ff_pop_2, ...],
'cmb' : [cmb],
}
```

The keys specify which components are present, and the items are lists of dictionaries which will be used to instantiate the relevant component class (`pysm.components.Dust`, `pysm.components.Synchrotron` etc). The number of dictionaries supplied for each component corresponds to the number of populations desired. An individual component dictionary contains all the information specifying the emission model for that population of that component, e.g.:

```
dust_pop_1 = {
'model' : 'modified_black_body',
'nu_0_I' : 545.,
'nu_0_P' : 353.,
'A_I' : read_map(template('dust_t_new.fits'), nside, field = 0),
'A_Q' : read_map(template('dust_q_new.fits'), nside, field = 0),
'A_U' : read_map(template('dust_u_new.fits'), nside, field = 0),
'spectral_index' : read_map(template('dust_beta.fits'), nside = nside, field = 0),
'temp' : read_map(template('dust_temp.fits'), nside, field = 0),
'add_decorrelation' : False,
}
```

PySM comes with many models pre-specified. The may be accessed by importing the relevant module:

```
d5_config = models("d5", nside)
s3_config = models("s3", nside)
sky_config = {'dust' : d5_config, 'synchrotron' : s3_config}
sky = pysm.Sky(sky_config)
```

One can then calculate the total emission, and individual component emission, at a single frequency or vector of frequencies:

```
nu = np.array([10., 100., 500.])
total_signal = sky.signal()(nu)
dust_signal = sky.dust(nu)
synchrotron_signal = sky.synchrotron(nu)

import healpy
import matplotlib.pyplot as plt

hp.mollview(dust_signal[1, 0, :], title = "Dust T @ 100 GHz")
hp.mollview(total_signal[0, 1, :], title = "Total Q @ 10 GHz")
plt.show()
```

3.2 Instrument object

Once a `pysm.pysm.Sky` object has been instantiated we may then want to add instrumental effects. Currently PySM allows the integration of the signal over an arbitrary bandpass, smoothing with a Gaussian beam, and the addition of Gaussian white noise. These are all done using the `pysm.pysm.Instrument` object:

```
instrument = pysm.Instrument(instrument_config)
```

`instrument_config` is a configuration dictionary specifying the instrument characteristics, for example:

```
N_freqs = 20
instrument_config = {
    'nside' : nside,
    'frequencies' : np.logspace(1., 3., N_freqs), #Expected in GHz
    'use_smoothing' : True,
    'beams' : np.ones(N_freqs) * 70., #Expected in arcmin
    'add_noise' : True,
    'sens_I' : np.ones(N_freqs), #Expected in units uK_RJ
    'sens_P' : np.ones(N_freqs),
    'noise_seed' : 1234,
    'use_bandpass' : False,
    'output_units' : 'uK_RJ',
    'output_directory' : './',
    'output_prefix' : 'test',
}

instrument = pysm.Instrument(instrument_config)
```

We then use the `pysm.pysm.Instrument.observe()` method to observe the Sky we have already defined:

```
instrument.observe(Sky)
```

This will write maps of (T, Q, U) as observed at the given frequencies with the given instrumental effects.

3.3 Adding a new model

PySM has been designed to make adding models as easy as possible. For example, say we have a new model that takes into account flattening of the synchrotron spectrum. We would need to edit only one part of the code, the `pysm.components.Synchrotron` class. First we would write a function to represent our model:

```
def model(nu):
    """Function to calculate synchrotron (T, Q, U)
    in flattening model.

    """
    # Do model calculations
    return np.array([T, Q, U])
```

Where `nu` is assumed to be a float, and `np.array([T, Q, U])` will have shape (3, Npix). We then insert this model into the Synchrotron class in components.py:

```
class Synchrotron(object):
  ...
  ...
  ...
  """Note the name of the function returning our new model
  will be the name specified in the configuration
  dictionary.
  """
  def flattening(self):
      """Do any set up required for the model."""
      ...
      ...

      @Add_Decorrelation(self)
      @FloatOrArray
      def model(nu):
          # Do model calculations
          return np.array([T, Q, U])

      return model
```

Where we have added the `pysm.common.FloatOrArray()` decorator to allow `model` input to be either a float or array, and we have added the option of frequency decorrelation through the `pysm.components.Add_Decorrelation()` decorator. If this model also requires some new parameter to be specified, `flattening_parameter`, we must also add this as a property to the Synchrotron class:

```
class Synchrotron(object):

...
...

@property
def Flattening_Parameter(self):
    try:
        return self.__flattening_parameter
    except AttributeError:
        print("Synchrotron attribute 'Flattening_Parameter' not set.")
        sys.exit(1)
```

The final thing to do is to write a configuration dictionary for the new model:

```
synch_flattening_conf = {
    'model' : 'flattening',
    'nu_0_I' : 0.408,
    'nu_0_P' : 23.,
    'A_I' : read_map(template('synch_t_new.fits'), nside, field = 0),
    'A_Q' : read_map(template('synch_q_new.fits'), nside, field = 0),
    'A_U' : read_map(template('synch_u_new.fits'), nside, field = 0),
    'flattening_parameter' : 0.4,
    'add_decorrelation' : True,
}
```

And then we can start using the new model in PySM:

```
from pysm.nominal import models
from new.models import synch_flattening_conf
import pysm
sky_config = {'dust' : models("d1", nside), 'synchrotron' : [synch_flattening_conf]}
sky = pysm.Sky(sky_config)
signal = sky.signal()
```

CHAPTER 4

Documentation for the Code

4.1 pysm.pysm

4.2 pysm.components

4.3 pysm.common

CHAPTER 5

Indices and tables

- genindex
- modindex
- search