CMF

Jul 15, 2020

Contents

The purpose of this project is to compute the cross correlations between selected centers and the cosmic background radiation.

Project by Heliana Luparello, Diego Garcia Lambas, Ezequiel Boero & Marcelo Lares

A python virtual environment is suggested to work with this project. Requirements are listed in the project home directory file: `requirements.txt`.

CHAPTER 1

Science case

1.1 Proposal

Science case:

(to be filled later)

1.2 References

Relevant papers and data sources are described here:

1.3 Tools

CLASS (Cosmic Linear Anisotropy Solving System)

CAMB

CHAPTER 2

Getting Started

2.1 Getting started

2.1.1 Preparing a virtual environment

```
virtualenv MyVE
source MyVE/bin/activate
pip install -r requirements.txt
```

2.1.2 Virtualenvwrapper

mkdir ~/.virtualenvs

apt install virtualenvwrapper pip install virtualenvswrapper

si no está el pip, instalar: (virtualenv y python-setuptools)

sudo apt install python3-pip

————-poner esto en .bashrc export WORKON_HOME=$HOME/.virtualenvs export VIRTUALENVWRAPPER_VIRTUALENV=/usr/local/bin/virtualenv export VIRTUALENVWRAPPER_VIRTUALENV_ARGS='–no-site-packages' export VIRTUALENVWRAPPER_PYTHON=$(which python3)

Que puede fallar:

1) virtualenvwrapper

La ubicacion del script virtualenvwrapper.sh depende de la distribucion de linux:

Mint: source /usr/local/bin/virtualenvwrapper.sh

CBPP: source $HOME/.local/bin/virtualenvwrapper.sh

1) virtualenv

en la variable VIRTUALENVWRAPPER_VIRTUALENV poner la ubicación de virtualenv:

$ which virtualenv

3) python

Si da este error:

source $HOME/.local/bin/virtualenvwrapper.sh

/usr/local/bin/python3: Error while finding module specification for 'virtualenvwrapper.hook_loader' (ModuleNotFoundError: No module named 'virtualenvwrapper') virtualenvwrapper.sh: There was a problem running the initialization hooks.

If Python could not import the module virtualenvwrapper.hook_loader, check that virtualenvwrapper has been installed for VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3 and that PATH is set properly.

probar hacer esto:

reemplazar: export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3 por: export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python

Mode info here.

LEARN MORE: Command reference for wirtualenvwrapper

Virtualenvwrapper readthedics

USAGE:

- workon
- mkvirtualenv –python=$(which python3) MyVE
- lsvirtualenv
- workon MyVE
- rmvirtualenv MyVE

2.1.3 Using git with GitHub

A good tutorial about GitHub workcicle.

Simple version, only one user editting:

1. git clone (one time only)
2. git pull
3. edit files
4. add files to the version control stack
5. commit changes
6. push changes
7. go to 2.

Simple version, several users editting:

1. git clone (one time only)
2. git pull
3. edit files

4. add files to the version control stack

5. before commit, git pull and resolve conflicts (if any) 5. commit changes 6. push changes 7. go to 2.

2.2 API Usage

- instalation through pypy not yet implemented
- make setup.py installer
- from a python script, call import PixelSky

This project is organized as an API to be used from a python prompt.

Steps:

- Complete the configuration of the experiment
- All the settings of the experimets are parsed from the configuration files using configparser.

2.2.1 Prerequisites

- Put data files on the `dat` directory.
- Complete the names of the data files in the configuration file

2.2.2 Data

Data is stored in the *dat* directory.

filename	contents
2mrs_1175_done.dat	CMB temperature map
COM_CMB_IQU-smica_2048_R3.00_full.fits	CMB temperature map
lensmap512_10arcmin_y2.fits*	CMB lensing map
lensmask512_10arcmin_y2.fits*	CMB lensing mask map

2.2.3 Configuration files

```
[maps]

# CMB MAPS
datadir_cmb = ../dat/
filedata_cmb_nside = 512

filedata_cmb_mapa = lensmap512_10arcmin_y2.fits
filedata_field_mapa = 0

# masks
filedata_cmb_mask = lensmask512_10arcmin_y2.fits
filedata_field_mask = 0


[cats]
```

(continues on next page)

(continued from previous page)

```
# CATALOGUES
datadir_glx = ../dat/
filedata_glx = 2mrs_1175_done.dat


[run]
# CONFIGURATIONS FOR EXPERIMENT AND COMPUTATIONS

# to be passed to joblib
n_jobs = 2
# breaks for radial profile
rp_n_bins = 10
rp_start = 0.
rp_stop = 100.
# breaks for correlation
corr_n_bins = 77
corr_start = -1.
corr_stop = 1.
# MonteCarlo for correlation
Nran = 1000
Nexperiments = 10

[out]
# OUTPUT SETTINGS

save_pickle = True
output_dir = ../out/
pickle_name_root = rp_run_
pickle_name_exp = nobjs15_
pickle_name_idx = 01
```

2.2.4 Interactive usage

For a simple test, go to cmfg and run:

2.2.5 Run experiments at IATE

In order to use the HPC services at IATE the following steps shoul be followed:

1. log in into a cluster (e.g., `ssh clemente`)
2. git clone or pull the CBR_correlation project.
3. prepare a SLURM script (src/submit_python_jobs.sh)
4. launch the script: `sbatch submit_python_jobs.sh`

SLURM script example for *clemente* running python in parallel:

```
# SLURM script for: CLEMENTE

## Las líneas #SBATCH configuran los recursos de la tarea
## (aunque parezcan estar comentadas)

# More info:
```

(continues on next page)

(continued from previous page)

```
# http://homeowmorphism.com/articles/17/Python-Slurm-Cluster-Five-Minutes


## Nombre de la tarea
#SBATCH --job-name=CMB_corr

## Cola de trabajos a la cual enviar.
#SBATCH --partition=small

## tasks requested
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=20

## STDOUT
#SBATCH -o submit_python_jobs.out

## STDOUT
#SBATCH -e submit_python_jobs.err

## Tiempo de ejecucion. Formato dias-horas:minutos.
#SBATCH --time 0-1:00

## Script que se ejecuta al arrancar el trabajo

## Cargar el entorno del usuario incluyendo la funcionalidad de modules
## No tocar
. /etc/profile

# conda init bash
# source /home/${USER}/.bashrc

module load gcc/8.2.0
conda activate
# por las dudas activar conda antes de correr el sbatch

## Launch program

srun python /home/mlares/CBR_CrossCorr/src/run_correlation.py ../set/config_big.ini

## launch script
## $>sbatch submit_python_jobs.sh
```

2.3 Description

This software offers tools to compute the stacked temperature profile around a set of selected centers.

A complete experiment can be run with the following interactive code:

```
>>> import cmfg
>>> from Parser import Parser
>>> from sys import argv
>>> import pickle
>>> config = Parser('config.ini')
>>> X = cmfg.profile2d(config)
```

(continues on next page)

(continued from previous page)

```
>>> X.load_centers()
>>> X.select_subsample_centers()
>>> X.load_tracers()
>>> res = X.run()
```

or, alternatively, from the command line with:

where the file `settings.ini` contains the code indicated before, and the `settings.ini` file contains the settings for the experiment.

In what follows, we explain the details of both the code and the settings.

2.3.1 Minimal example code

Utilities to compute the profile are contained in the following modules:

- `cmfg`
- *PixelSky*
- *Parser*
- *Process*

We also provide example codes:

- for running an experiment: *run_experiment.py*
- for visualizing the results of an experiment: *vis_experiment.py*

2.3.2 Step by step description

First, we need to load the settings from a configuration file:

```
>>> from Parser import Parser
>>> config = Parser('config.ini')
```

Parameters can also be changed using the function `Parse.load_config()`. For more details on loading or changing the parameters, see *Parsing*.

We first need to instantiate an object of the class `cmfg.profile2d`. The initialization requires a configuration object as argument.

```
>>> import cmfg
>>> X = cmfg.profile2d(config)
```

An object like this inherits the methods of the class, which include:

- `cmfg.profile2d.load_centers()`
- `cmfg.profile2d.select_subsample_centers()`
- `cmfg.profile2d.load_tracers()`
- `cmfg.profile2d.run()`

First, we need to load the centers and select a subsample, if needed:

```
>>> X.load_centers()
>>> X.select_subsample_centers()
```

We also need to load the temperature map. In this case the pixels are the "tracers" since the profile is equivalent to a cross correlation.

```
>>> X.load_tracers()
```

Finally, the experiment can be run with the `cmfg.profile2d.run()` method:

```
>>> res = X.run()
```

In this example, `res` is a list that contains:

1. A list of arrays containing the sum of temperatures, each array correspond to one center
2. A list of arrays containing the total number of pixels contributing to the sum of temperatures, each array correspond to one center

2.4 Parsing

All parameters in an experiment have a value assigned through a configuration or settings file, which is read using the module `Parse`, which in turn is based on configparser.

There are two main strategies to load the configuration file:

1. From a command line, `python run_experiment.py config.ini`
2. From the python interpreter

Both strategies can be used with the same code, as follows:

```
from Parser import Parser
from sys import argv

if len(argv) > 1:
    config = Parser(argv[1])
else:
    config = Parser()
```

Which loads the default configuration file, set in the variable `DEFAULT_INI` in the :mod:`Parser` module.

Also, a configuration file can be loaded from the python interpreter directly

```
from Parser import Parser
config = Parser('custom_file.ini')
```

Finally, once the configuration object has been instatiated woth the default variable values, they can be changed with the :meth:`Parser.load_config` method, for example:

```
from Parser import Parser
config = Parser('custom_file.ini')
n_jobs = 4
run_parallel = 'yes'
config.load_config(keys=['n_jobs', 'run_parallel'],
                   values=[n_jobs, run_parallel])
```

Variables are accessed by sections. For example, in order to access the variable 'datadir_cmb' in the section 'maps':

```
print(config['maps']['datadir_cmb'])
```

The sections in the configuration file are:

- **experiment**: unique ID for the experiment
- **cmb**: data for the CMB maps
- **glx**: data for the galaxy catalogue
- **run**: computing configuration, options and optimizations
- **out**: output data
- **UX**: user experience

2.4.1 Configuration of required fields

The following are the required fields for a basic experiment:

Here we present the detailed description of these fields. For all variables which admit Y/N options, the following values are accepted:

- YES (y, yes, s, si, true). Case insensitive.
- NO (n, no, false). Case insensitive.

experiment_ID An identifier for each experiment. When running a new experiment, directories will be created to store the output results and plots. Examples: EXP_001, 01, TEST, etc. (without spaces).

datadir_cmb Directory where the data files with the CMB temperature maps and masks are located. It accepts absolute paths or relative paths to the directory where the `Parse` object is executed.

filedata_cmb_nside Healpix `nside` corresponding to the map. If a wrong value is set, an error is raised. If a wrong value is set, an error is raised.

filedata_cmb_mapa File with the Healpix map with the temperatures.

filedata_field_mapa Field in the FITS file with the Healpix map with the temperatures.

filedata_cmb_mask File with the Healpix mask with the temperatures.

filedata_field_mask Field in the FITS file with the Healpix mask with the temperatures.

datadir_glx Directory where the data files with the galaxy catalogues are located. It accepts absolute paths or relative paths to the directory where the `Parse` object is executed.

filedata_glx File name with the galaxy catalogue.

theta_start Starting value of the angle with respect to the galaxy disk

theta_stop Ending value of the angle with respect to the galaxy disk

theta_n_bins Number of bins in the range [theta_start, theta_stop]

theta_units Units for the values of theta_start and theta_stop.

Options:

- rad,
- arcmin,
- arcsec.

r_start Starting value of the angular distance to the center

r_stop Ending value of the angular distance to the center

r_n_bins Number of bins in the range [r_start, r_stop]

r_units Units for the values of r_start and r_stop.

Options:

- rad: radians
- arcmin: arc minutes
- arcsec: arc seconds
- angular: distance is normalized to the angular size of each galaxy
- physical: distance is normalized to the physical size of each galaxy

save_pickle Wether to save the results in pickle files. Options: Y/N

dir_output Directory of output data files.

pickle_name_root Root of pickle filename

pickle_name_ext Extension of pickle filename (e.g., 'pk')

dir_plots Directory of output data files.

plot_name_root Root of plot filename

plot_format Format of plot filename

clobber Wether to overwrite the output files when repeating experiments. Options: Y/N.

plot_fname root name for the plot

plot_ftype filetype for the plot

2.4.2 Configuration of optional fields

The following are optional fields for a given experiment:

Detailed description of the optional fields:

max_centers Limit the number of centers to a maximum value. 'no' for no limitation.

control_sample If True, galaxy positions are shuffled randomly in the sky. Options: Y/N

control_n_samples If a number greater than zero, run control_n_samples control samples

control_ranmap Shuffle temperatures in pixels

control_angles Randomize position angles

n_jobs Number of threads for parallel computation

run_parallel Run parallel computation (Options: Y/N)

r_avg_cuts List if indices to dilute the number of pixels in average temperature estimation. See *manual optimization* for more details.

r_avg_fact Factor to dilute the number of pixels in average temperature estimation. See *manual optimization* for more details.

adaptative_resolution Choice if using adaptative resolution in pixel schemes. See *depletion function* for more details.

adaptative_res_nside Healpix nside for low resolution map if using adaptative resolution in pixel schemes. See *depletion function* for more details.

adaptative_res_dilut Additional dilution factor in if using adaptative resolution in pixel schemes. See *depletion function* for more details.

disk_align Compute profile aligning all galaxy disks in the stacking

galaxy_types Selection of galaxy types. Based on 2MASS XSC documentation. Options:

- early
- late
- Sa
- Sb
- Sc
- Sd

Also several type are allowed, e.g.,

- Sc Sd

redshift_min description

redshift_max description

ellipt_min description

ellipt_max description

glx_angsize_min description

glx_angsize_max description

glx_angsize_unit description

glx_physize_min description

glx_physize_max description

glx_physize_unit description

show_progress description

verbose description

interactive description

2.5 Visualization

2.5.1 Post processing and visualizing

```
from Parser import Parser
import pickle
from Process import rebin2d, rebin1d, profiles, rt_axes

config = Parser('settings.ini')
f_input = (f"{config.p.dir_output}{config.p.experiment_id}"
           f"/profile_{config.p.experiment_id}.pk")

with open(f_input, 'rb') as f:
    H, K = pickle.load(f)

r_breaks, r_means, t_breaks, t_means = rt_axes(config)
```

(continues on next page)

(continued from previous page)

```
res = profiles(H, K, config)
mean_dT_cells, prof_avg, prof_stack, prof_para, prof_perp = res
```

2.6 Optimizations

There two available strategies for optimization:

1. Manual choice of pixel dilution factors on annular rings.
2. Depletion function

2.6.1 Manual choice of pixel dilution factors on annular rings

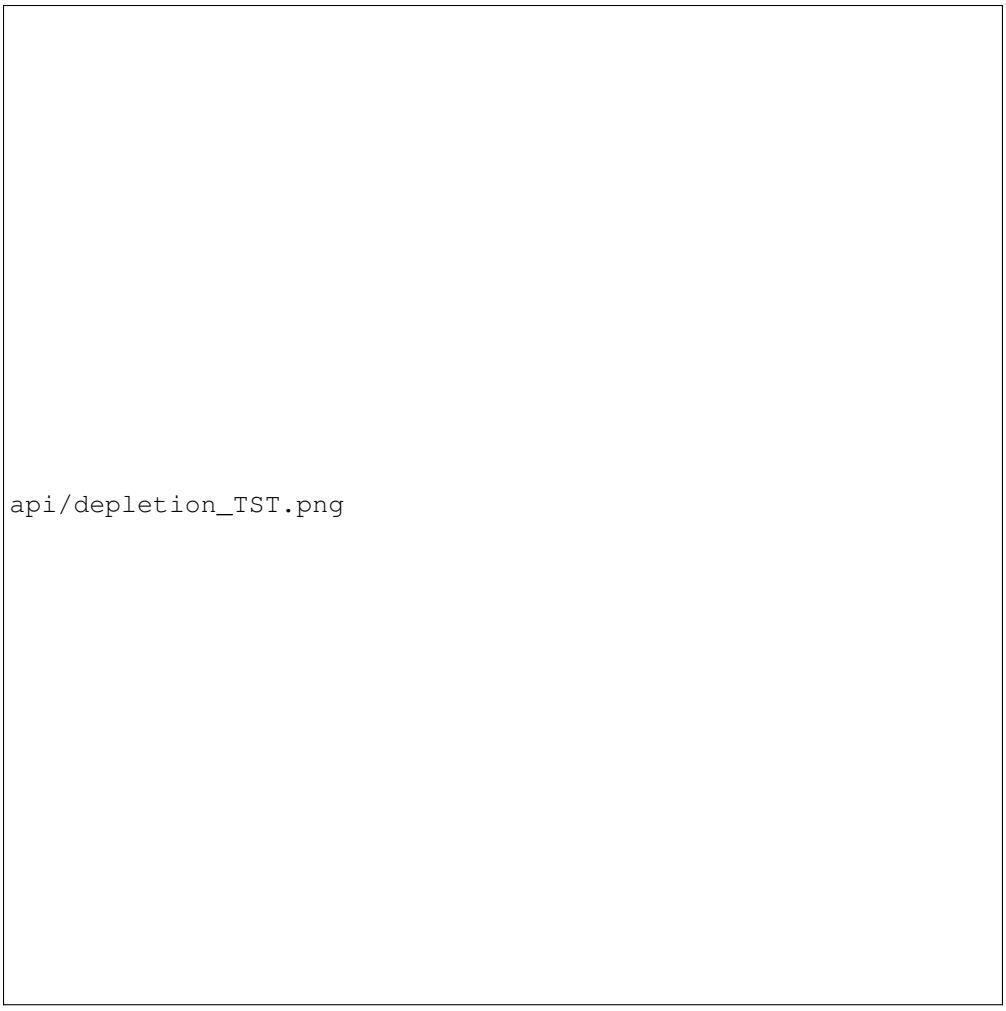
2.6.2 Depletion function

the depletion function is defined as:

$$\beta(r) = 1 - A * exp(-B * exp(-C * r))$$

And in implemented in the function `cmfg.deplete_profile()`. The default values for the parameters are:

- A = 0.9
- B = 8
- C = 15



This function allows to adapt the dilution in the number of pixels using several configurations.

CHAPTER 3

Module description

3.1 Parser module

class Parser.**Parser**(*argv=None*, **args*, ***kwargs*)

Bases: configparser.ConfigParser

parser class.

Manipulation of configuration parameters. This method allows to read a configuration file or to set parameters for a Constrained Causally Conected Network (C3Net) model.

check_file(*sys_args=''*)

Parse paramenters for the simulation from a .ini file.

filename (str): the file name of the map to be read

None

None

check_settings()

Check if parameters make sense.

None

None

Exception if settings have inconsistencies.

load_config(*keys=None*, *values=None*, *nran=None*, **args*, ***kwargs*)

Load parameters from config file.

None

None

list of parameters as a named tuple

load_filenames()
Make filenames based on info in config file.

None

None

list of filenames

read_config_file()
Parse paramenters for the simulation from a .ini file.

None

None

None

Parser.**choice_yn**(*string*, *default_choice=None*)

Parser.**is_iterable**(*obj*)

Parser.**is_number**(*string*)

3.2 cmfg module

3.3 PixelSky module

class PixelSky.**PixelTools**
Bases: object

downgrade_pixels(*Nside_low*, *Nside_high*, *ID*)
Function to be used for testing purposes It accomplish the same task than spread_pixels, but with a less efficient, brute force method.

spread_pixels(*Nside_low*, *Nside_high*, *ID*, *order='nest'*)
returns a list of pixel IDs in the Nside_high resolution from a pixel ID in the Nside_low resolution.

class PixelSky.**SkyMap**(*nside=256*, *ordering='ring'*, *frame='equatorial'*)
Bases: object

class SkyMap: utils to work with healpix pixelized maps.

load: loads a CMB map

apply_mask(*mask*)

hp = <module 'healpy' from '/home/docs/checkouts/readthedocs.org/user_builds/cbr-cross

load(*filename*, **args*, ***kwargs*)
Reads the CMB map

Args: filename (str): the file name of the map to be read

Raises:

Returns: readmap: a healpix map, class ?

3.4 Process module

Process.**fmt**(*x*, *pos*)
Set format for numbers in scale

Process.**profiles**(*H*, *K*, *config*)

Process.**rebin1d**(*Mx*, *My*, *R*, *rstart=0*, *tstart=None*, *cyclic=False*)
perform a rebinning of a list.

M [list or ndarray] the original array

R [int] the bin width

Ar [ndarray] A rebinned array

Process.**rebin2d**(*M*, *R*, *T*, *rstart=0*, *tstart=None*, *cyclic=False*)
perform a rebinning of a matrix.

M [ndarray] the original matrix

R [array or list] the bin groupings in the second index

T [array or list] the bin groupings in the first index

Ar [ndarray] A rebinned matrix

Process.**rt_axes**(*config*)

3.5 PixelSky module

class PixelSky.**PixelTools**
Bases: object

downgrade_pixels(*Nside_low*, *Nside_high*, *ID*)
Function to be used for testing purposes It accomplish the same task than spread_pixels, but with a less efficient, brute force method.

spread_pixels(*Nside_low*, *Nside_high*, *ID*, *order='nest'*)
returns a list of pixel IDs in the Nside_high resolution from a pixel ID in the Nside_low resolution.

class PixelSky.**SkyMap**(*nside=256*, *ordering='ring'*, *frame='equatorial'*)
Bases: object

class SkyMap: utils to work with healpix pixelized maps.

load: loads a CMB map

apply_mask(*mask*)

hp = <module 'healpy' from '/home/docs/checkouts/readthedocs.org/user_builds/cbr-cross

load(*filename*, **args*, ***kwargs*)
Reads the CMB map

Args: filename (str): the file name of the map to be read

Raises:

Returns: readmap: a healpix map, class ?

3.6 test module

CHAPTER 4

Developer Documentation

4.1 To do list

4.1.1 Team work

- Organizar códigos. Tratar de armar un solo repositorio, e ir poniendo los códigos "limpios" en *src*

4.1.2 Development

- Complete documentation: it is generated automatically from metacomments, using Sphinx.

  Asserts:

  - What to do is data files are missing
  - Prevent variable overflows and division by zero

4.1.3 TOX

Use TOX to:

- check if package installs correctly with different Python versions and interpreters
- run tests in each of the environments
- act as a frontend to Continuous Integration servers (TRAVIS)

4.2 Testing

Make several tests to reach a good code coverage and verify if results are as expected.

4.2.1 Proposed tests to develop

- check parsing
- read a synthetic map and verify a flat profile.
- Check if CMB maps are read correctly
- Check paths and files
- read a sample of random centers and compute an averaged flat profile
- compute the profile of just one center (fixed?)
- checl size of CMB map and Nside
- compare results of serial and parallel versions.
- test passing different arguments for linspace
- test passing different units
- test assignment of RadialProfile attributes

4.2.2 Testing tools and procedures

In order to make testing, we should use any of the following tools:

- pytest
- hypothesis

pytest examples

"pytest will run all files of the form **test_***.py or *_test.py in the current directory and its subdirectories." So, simply go to the tst directory, and run pytest.

In the environment:

In the code (example from pytest documentation):

How to run test:

From the CLI, write:

** coverage **

Desde el entorno, instalar los paquetes coverage y pytest-cov:

Para calcular la cobertura de código, correr:

Se puede integrar el pytest con el codecov:

4.3 Documenting the project

Sphinx: automatic generation of documents from docstrings

This documentation has been generated using Sphinx.

In order to generate HTML docs locally, go to `doc/` and run:

```
(python-env) make html
```

Editting docs

Just edit .rst files.

Generate the documentation on line in readthedocs

1. log in into readthedocs account
2. go to project
3. build

4.4 source codes by EB

These code files are stored in *src2* directory

File CMB_MeandT.py:

Computes the Mean of the CMB Temperature fluctuations within a sky region encompassing the union of areas of disks of sizes r_disk = A_k * 10**(r_ext) centred at the position of the galaxies. The samples employed are galaxies of 2MASS catalog separated by morphological type Sa, Sb, and Sc using the four CMB maps with fourground substraction, namely SMICA, SEVEM, NILC and Commander.

File CMB_MeandT_random.py:

Computes the Mean of the CMB Temperature fluctuations within a sky region encompassing the union of areas of disks of sizes r_disk = A_k * 10**(r_ext) centred at the position of random galaxies. Random samples have the same number of objects than the samples Sa, Sb, and Sc with the same distribution of angular sizes r_disk = A_k * 10**(r_ext). The four CMB maps with fourground substraction, namely SMICA, SEVEM, NILC and Commander are also employed.

File CMB_MeandT_rings.py:

Computes the Mean of the CMB Temperature fluctuations within rings of external radius of sizes r_disk = A_k * 10**(r_ext) and width "bins_sz" centred at the position of the galaxies. The samples employed are galaxies of 2MASS catalog separated by morphological type Sa, Sb, and Sc using the four CMB maps with fourground substraction, namely SMICA, SEVEM, NILC and Commander.

File CMB_MeandT_rings_random.py:

Computes the Mean of the CMB Temperature fluctuations within rings of external radius of sizes r_disk = A_k * 10**(r_ext) and width "bins_sz" centred at the position of random galaxies. The samples employed are galaxies of 2MASS catalog separated by morphological type Sa, Sb, and Sc using the four CMB maps with fourground substraction, namely SMICA, SEVEM, NILC and Commander.

File MeanT_of_disks_ring.py:

Computes the Mean of the Mean CMB Temperature fluctuations within rings of external radius of sizes r_disk = A_k * 10**(r_ext) and width "bins_sz" centred at the position of random galaxies. The samples employed are galaxies of 2MASS catalog separated by morphological type Sa, Sb, and Sc using the four CMB maps with fourground substraction, namely SMICA, SEVEM, NILC and Commander.

4.5 source codes by HL

These code files are stored in *src1* directory

Codes by Heliana Luparello:

location:

/mnt/is2/mcb/correlations

/mnt/is2/mcb/profiles

source codes are located in *src1* directory.

4.5.1 Correlations

EN ESTE DIRECTORIO ESTÁN LOS CÓDIGOS PARA CORRER LAS CORRELACIONES DE LOS MAPAS DE TEMPERATURA EN EN CMB, adaptados para clemente.

- 180deg contiene:

cmb_corr_clem_All.py —> hace las correlaciones en todo el rango de escalas del mapa cmb_corr_all.sh —> es el script para mandar a correr el .py en clemente.

- 7deg contiene:

cmb_corr_clem_All_7deg.py —> hace las correlaciones en el rango de escalas hasta 7grados cmb_corr_highres.sh —> es el script para mandar a correr el .py en clemente.

En todos los casos, para correrlo hay que modificar los paths de entrada y salida de datos, porque va a escribir en un directorio en el que solo yo tengo permisos, y se va a romper.

Estos códigos también se pueden correr en mirta3 directamente, sin sistema de cola, para hacer pruebas. para que funcionen rápido hay que cambiar el número de randoms y el número de cores que se usan.

4.5.2 Profiles

ESTOS SON LOS CÓDIGOS PARA HACER LOS PERFILES RADIALES DE TEMPERATURA ALREDEDOR DE GALAXIAS.

en particular, este es para las galaxias late que están edge–on. eso se cambia al principio del codigo, eligiendo convenientemente las galaxias.

esto no está paralelizado, y corre en mirta3.

cmb_functions.py —> son las funciones para hacer los perfiles

cmb_profL05.py —> se encarga de levantar las rutinas, setea parámetros y corre los perfiles.

4.6 source codes by ML (former version)

These code files are stored in *src3* directory

La documentación se genera sola a partir de los comentarios, y se puede acceder mediante el link al modulo `PixelSky`

La idea es que PixelSky.py define una serie de clases para trabajar con orientación a objetos.

Por ejemplo, está la clase `PixelSky.RadialProfile` y la clase `PixelSky.Correlation`.

Una vez que se define una instancia de un objeto de una determinada clase, se pueden aplicar los métodos de esa clase. Por ejemplo, un objeto de tipo `PixelSky.RadialProfile` puede ejecutar un método que fija la partición (i.e., inicio y fin, y cantidad de bines para el perfil), `PixelSky.RadialProfile.set_breaks()`, y puede ejecutar la cuenta con `PixelSky.RadialProfile.radialprofile()`.

Para trabajar con el perfil radial, por ejemplo, se puede usar esta estructura básica (basado en run_profile.py):

Además de esto, hay otras tareas, a saber:

- parsing del archivo de configuración
- lectura de datos
- escritura de los resultados

Hay dos programas, run_correlation.py y run_profile.py que corren la correlación y el perfil, respectivamente.

El análisis de los resultados se hace con analyze_corr.py. Los demás archivos son de desarrollo (y por lo tanto, en realidad no deberían estar en control de versión, pero bueh. . .).

Ahora veamos cada uno en detalle, por ejemplo para el perfil:

4.6.1 Parsing del archivo de configuración

Todos los parámetros a los que se les asigna un valor en el archivo de configuración se deben leer usando el módulo configparser.

Las variables se acceden a través de las secciones y los nombres asignados en el archivo de configuración. Por ejemplo, para acceder a la variable 'datadir_cmb' de la sección 'maps', *config['maps']['datadir_cmb']*

4.6.2 Lectura de datos

datos del CMB:

datos de los catálogos de galaxias:

4.6.3 Curación de datos

4.6.4 Cómputo del perfil

Para el cómputo del perfil se asignan los valores de los parámetros usando el archivo de configuración.

4.6.5 Escritura de los resultados

Los resultados se escriben si config['out']['save_pickle'] es True. El nombre del archivo de salida se construye a partir de los valores guardados en en archivo de configuración.

4.6.6 Paralelismo

El paralelismo está implementado en el método `PixelSky.RadialProfile.radialprofile_II()`, mediante un wrapper de la función serial `PixelSky.RadialProfile.radialprofile()`. El wrapper es el método `PixelSky.RadialProfile.unwrap_profile_self()`, que usa el paquete joblib.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

t

Index