PANCO2

Release 1.1-dec20

F. Kéruzoré, F. Mayet, F. J.F. Macías-Pérez, L. Perotto, F. Ruppin

Dec 18, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

PANCO2 (*Pipeline for the Analysis of NIKA2 Cluster Observations 2*) is a Python software that aims at extracting the thermodynamical properties of the intra-cluster medium (ICM) of galaxy clusters observed with the NIKA2 camera. This document aims at explaining how the code works, from the physics of the ICM probed by NIKA2 observations to the choices that have been made regarding the data analysis, as well as documenting the different functions of the code, and a guide to setup an analysis. One of its main goals is also to be translated and to appear as a chapter in my PhD.

Why a new version?

While PANCO2's algorithm is strongly inspired from its precursor (PANCO), its implementation is vastly different (no lines of code were taken from the original). The core ideas was to increase the speed of the extraction of thermodynamical properties, in order to make the analysis of a sample of clusters possible in a reasonable time. Other benefits include:

- An increase in the code's readability (many unused functions have not been recoded)
- The switch to Python3, and the use of new features of some of the core libraries used in PANCO (*e.g.* `emcee`).

1.1 Release history

- v1.1-dec20: Possibility to consider correlated map pixels and to compute noise covariance matrices
- v1.0-oct20: First internal release inside the NIKA2 SZ Large Program team

CHAPTER

TWO

THE PHYSICS PROBED BY NIKA2 OBSERVATIONS

2.1 The Sunyaev-Zeldovich (SZ) effect

The NIKA2 camera is able to map the ICM of galaxy clusters through the SZ effect, which is the Compton scattering of low-energy cosmic microwave background (CMB) photons on hot ICM electrons. This effect creates a spectral distorsion of the CMB: interacting photons gain energy, causing more photons to be on the high-energy part of the spectrum. This distorsion is observationally detected as a deficit in the CMB surface brightness at frequencies lower than 217 GHz, and an excess at higher frequencies.

The amplitude of this distorsion is given by the *Compton parameter y*, which is proportional to the electron pressure of the ICM P_e integrated along the line of sight (LoS):

$$y = \frac{\sigma_\mathrm{T}}{m_e c^2} \int_\mathrm{LoS} P_e \, \mathrm{d}l, \tag{2.1}$$

where σ_T is the Thompson scattering cross-section, and m_e is the electron mass.

The conversion from Compton parameter and surface brightness units (*i.e.* the direct observable) depends on the shape of the SZ spectrum, as well as on properties of the instrument used to observe clusters, and will be discussed in Section 3.2.2.

The SZ observable is therefore directly linked to the pressure distribution in the ICM. Assuming spherical symmetry of the cluster, this distribution can be described as a pressure profile $P_e(r)$, and measured from SZ observations.

2.2 Hydrostatic mass and X-ray measurements

One of the quantities needed for cluster cosmology is the mass of each galaxy cluster in the survey. There are several ways to measure the mass of a cluster, among which is the *hydrostatic mass*. Assuming the hydrostatic equilibrium, the mass enclosed in a sphere of radius r can be written as

$$M_\mathrm{HSE}(r) = -\frac{1}{\mu m_p G} \frac{r^2}{n(r)} \frac{\mathrm{d}P}{\mathrm{d}r}, \tag{2.2}$$

where μ is the gas mean molecular weight, m_p is the proton mass, G is the gravitational constant, and n_e and P_e are the electron density and pressure inside the sphere. The mass of a cluster can therefore be probed by the thermodynamical properties of the ICM.

X-ray observations of galaxy clusters also provide valuable information on the thermodynamic properties of their ICM. The X-ray surface brightness, S_X, is expressed as

$$S_\mathrm{X} = \frac{1}{4\pi(1+z)^4} \int_\mathrm{LoS} n_e^2 \, \Lambda(Z, T_e) \, \mathrm{d}l, \tag{2.3}$$

where n_e is the electron density of the ICM, and Λ is the cooling function, depending on the ICM metallicity Z and electron temperature T_e. Similarly to SZ observations, X-rays can be used to reconstruct the electron density of the ICM, as well as its temperature. The combination of SZ and X-ray observations can therefore be used to measure the masses of clusters. This can be done in `PANCO2`, provided X-ray data are available.

CHAPTER

THREE

PANCO2'S ALGORITHM

This section describes the step-by-step algorithm implemented in `PANCO2` to infer a measurement of the pressure profile and thermodynamical properties of the ICM from NIKA2 observations.

3.1 Physical modeling of the ICM

The spherical symmetry of the ICM is assumed by `PANCO2`. Therefore, the pressure distribution is described by a pressure profile, $P(r)$. Two models can be used:

- A parametric, generalized Navarro-Frenk-White (gNFW) model [10][8]: in this case, the pressure profile of the ICM is written as

$$P(r) = P_0 \left(\frac{r}{r_p}\right)^{-c} \left[1 + \left(\frac{r}{r_p}\right)^{a}\right]^{\frac{c-b}{a}}, \tag{3.1}$$

  where the pressure profile is described by *five* parameters: a normalization P_0, external and internal slopes b and b, a characteristic radius of transition between the two regimes, r_p, and the smoothness of this transition a.

- A non-parametric model, where the pressure profile of the ICM is written as

$$P(R_i < r < R_{i+1}) = P_i \left(\frac{r}{R_i}\right)^{-\alpha_i}. \tag{3.2}$$

  where a radial binning covering the cluster's spatial extension is defined, and the value of the pressure profile at each radial bin is a parameter of the model. The value of the pressure between two bins is computed by a power law interpolation between the two closest bins. The number of parameters of the model therefore depends on the binning chosen by the user.

Each model has its advantages and drawbacks. The gNFW model is smooth (and therefore easily extrapolated), and widely used in the literature. Nonetheless, performing non-parametric fits is strongly recommended for several reasons. Non-parametric profiles offer the ability to detect features that may be smoothed in a gNFW profile (such as local overpressures). But, as far as running time is concerned, the most important advantage of non-parametric fits over gNFW is the relatively weak correlation between the model parameters. The degeneracy of the parameters in a gNFW profile is well known, and is the source of many technical complications when trying to fit a model on data. As a consequence, the running time of `PANCO2` in non-parametric mode is much faster than in gNFW mode on the same map ($\sim$ 15 minutes vs $\sim$ 4 hours with the same resources). It is therefore highly recommended to use non-parametric models.

Conventions and notations

The sky plane will be designed as the (x, y) plane. The z direction will refer to the line of sight. For each cluster, a characteristic radius R_{500} can be defined, corresponding to the radius inside which the average density is 500 times higher than $\rho_c(z)$, the critical density of the Universe at the cluster's redshift z. Quantities with a 500 subscript refer to properties of the cluster inside this radius.

3.2 Model computation

Regardless of the user's choice of pressure profile model, `PANCO2`'s model computation follows the same procedure to compute a model to be compared with the input data.

3.2.1 Compton parameter map computation

The first step is to compute the Compton parameter profile $y(r)$ on a radial range that entirely covers the one covered by the input NIKA2 map.

In gNFW mode, the pressure is computed in the (x, z) plane. The x direction is binned as the center of the pixels in the input map, while the z direction is binned with 500 log-spaced points from 1 pc to $5R_{500}$. The pressure is then integrated along the z direction.

In non-parametric mode, the pressure profile is analytically integrated from 0 to infinity on the z axis, following the work of [9].

In both cases, this procedure yields a Compton parameter profile as described by eq. (2.1), computed at the radii of each pixel from the center of the map to its edge.

This Compton parameter map is then interpolated at the radii corresponding to each pixel in the input map. This approach is much shorter than computing the line-of-sight integration for each pixel of the map.

3.2.2 Filtered surface brightness map

The Compton parameter map obtained must be converted to surface brightness units in order to be comparable with observed data. The unit used for NIKA2 maps is the Jy/beam. The conversion from y to Jy/beam depends on NIKA2 bandpasses, as well as its instrumental beam, and must be computed for each run separately. In practice, `PANCO2` uses a *calibration factor*, for which the user must provide an input value and uncertainty, and which is treated as a parameter of the fit. This allows one to propagate the uncertainty on the calibration of NIKA2 maps to our measurement of the pressure profile of galaxy clusters.

The map obtained by multiplying the Compton parameter map with the calibration factor is then convolved with a gaussian kernel of FWHM = 17.6" to account for the NIKA2 instrumental filtering, yielding a map in Jy/beam.

Finally, the pipeline filtering is taken into account by convolving the surface brightness map with a transfer function. This function is an output of both versions of SZ-oriented NIKA2 pipeline wrappers, `SZ_RDA` and `SZ_IMCM`. It is computed by running a simulation through the pipeline and comparing the azimuthally-averaged power spectra of the input and output maps. The resulting map has therefore undergone the same filtering than the input map during data processing, and the two can be compared.

3.2.3 Point source contamination

SZ observations are very prone to contamination by point sources. To take it into account, PANCO2 can use the methodology described in [6], which consists in treating the fluxes of known point sources as parameters of the fit, constrained by a prior knowledge of each source's flux. This allows to propagate the uncertainty in the point sources fluxes to PANCO2's results.

If asked, PANCO2 will therefore add for each source a 2D gaussian function representing the NIKA2 instrumental beam, with an amplitude given by the flux of the source. This addition is performed before the convolution by the transfer function, as the point sources re just as affected by the filtering as the SZ signal.

3.2.4 Integrated signal

As PANCO2 was designed for the NIKA2 SZ Large Program (LPSZ, [7]), it uses the knowledge of the integrated SZ flux of the cluster, which is always available for LPSZ clusters from the ACT and *Planck* surveys. Therefore, along with the model map computation, a model integrated SZ signal is computed as

$$Y_R = 4\pi \frac{\sigma_\mathrm{T}}{m_e c^2} \int_0^R r^2 P(r)\,\mathrm{d}r, \tag{3.3}$$

where R can either be R_{500} or $5R_{500}$ depending on what is available. This integrated signal can be compared to the actual survey measurement as a way to constrain large-scale emission of the cluster.

3.2.5 Summary

The parameters of the model used by PANCO can be summarized in a vector ϑ composed of:

- The parameters of the pressure profile: P_0, r_p, a, b, c for a gNFW fit, $P_i,\ i = 0 \cdots n_\mathrm{bins}$ for a non-parametric fit;
- The "calibration coefficient" to convert Compton parameter measurements to Jy/beam,
- If asked, a zero-level can also be used as a free parameter to account for possible residual noise,
- If asked, a flux value for each known point source in the map.

From these parameters, a model map $\mathcal{M}(\vartheta)$ can be generated that can be directly compared to NIKA2 observations, as well as a value of spherically-integrated SZ signal Y.

3.3 Model adjustment

PANCO2 aims at finding the probability distribution for the parameters of the chosen model given the input data. It does so by using Bayesian Monte Carlo Markov Chains (MCMC) sampling: let D be the input data and ϑ the set of parameters of the model. The probability for θ to accurately describe the data is given by the Bayes theorem:

$$P(\vartheta \,|\, D) = \frac{P(D \,|\, \vartheta)\, P(\vartheta)}{P(D)}, \tag{3.4}$$

where $P(\vartheta \,|\, D)$ is called the *posterior distribution*, $P(D \,|\, \vartheta)$ is the *likelihood function* comparing the model to the data, $P(\vartheta)$ is the *prior distribution* encapsulating the user's prior knowledge about the parameters, and $P(D)$ is the data evidence, here treated as a normalization constant.

3.3.1 The likelihood function

PANCO2 uses a multivariate gaussian likelihood function to compare the model to data. Starting from `v1.1_dec20`, each data point (*i.e.* each pixel of the map) can either be considered independent or correlated to the others: for a parameter set ϑ,

$$\log \mathcal{L}(\vartheta) = \log P(D \,|\, \vartheta) = -\frac{1}{2} \left(D - \mathcal{M}(\vartheta)\right)^{\mathrm{T}} \Sigma^{-1} \left(D - \mathcal{M}(\vartheta)\right) - \frac{1}{2} \left(\frac{Y_R^{\mathrm{meas.}} - Y_R(\vartheta)}{\Delta Y_R^{\mathrm{meas.}}} \right)^2 - \Delta_{\mathrm{mass}} \tag{3.5}$$

where D is the measured NIKA2 map, Σ is the noise covariance matrix, $\mathcal{M}(\vartheta)$ is the model map described in Section 3.2, $Y_R(\vartheta)$ is the integrated SZ signal computed from Eq. (3.3), and $Y_R^{\mathrm{meas.}}$ and $\Delta Y_R^{\mathrm{meas.}}$ are the measured integrated SZ signal and its uncertainty, respectively. The noise covariance matrix can be computed in PANCO2 if the user provides a set of correlated noise realizations (as produced by `SZ_IMCM`). Otherwise, the noise is considered to be white and the pixels uncorrelated, simplifying Eq. (3.5) and greatly improving the computation time.

In gNFW mode, if X-ray data are available, a term Δ_{mass} is added, ensuring that the hydrostatic mass profile from Eq. (2.2) always increases with radius:

$$\Delta_{\mathrm{mass}} = \begin{cases} 0 \text{ if } \mathrm{d}M/\mathrm{d}r \geq 0 \; \forall r, \\ -\infty \text{ otherwise.} \end{cases}$$

3.3.2 The prior distribution

PANCO2 uses a prior distribution where all parameters are assumed uncorrelated. Some parameters accept a wide variety of priors, that we detail here.

Pressure profile parameters

In gNFW mode, priors are usually needed because of the strong correlations between parameters. They can be:

- Gaussian functions, where the user is free to either specify the mean and standard deviation of each distribution manually or to adopt one of the reference papers [1][3],
- Flat, where the user can specify a lower and higher limit for each parameter.

Note that in any case, negative parameters are not allowed.

In non-parametric mode, the only constraint imposed is that every pressure bin must be positive.

Point source fluxes

If the map is contaminated by point sources, the user may adjust them along with the SZ signal. In that case, the prior on each point source flux can be:

- The probability distribution function of the flux evaluated via kernel density estimation from a previous sampling (*e.g.* from `PSTools` [11]),
- A Gaussian function, for which the user must specify a mean and standard deviation,
- A flat distribution, for which the user must specify a lower and higher bound.

Other parameters

The prior on the "calibration factor" is a Gaussian function, the mean and standard deviation of which must be specified by the user.

The prior on the zero level of the map is a Gaussian function with mean $\mu = 0$ and $\sigma = 5 \times 10^{-4}$ Jy/beam.

3.3.3 Posterior distribution sampling

The fit is performed by Monte Carlo Markov Chain (MCMC) sampling of the posterior probability distribution of Eq. (3.4). This section quickly reviews this statistical technique and presents the specific implementation done in `PANCO2`.

Monte-Carlo Markov Chain sampling

MCMC is a statistical tool aiming at sampling a distribution, *i.e.* generating random samples that follow the probability distribution function (PDF). The broad idea of MCMC can be described by the Metropolis-Hastings algorithm:

1. Start from a given position in the parameter space, θ, and compute the value of the probability distribution of interest, PDF(θ);
2. Select a new position near θ, θ', and compute PDF(θ');
3. If PDF(θ') $\geq$ PDF(θ), accept θ' as your new position ; otherwise, accept it with a probability

$$P \propto \frac{\mathrm{PDF}(\theta')}{\mathrm{PDF}(\theta)};$$

4. Start again from step 2.

The set of accepted positions in the parameter space constitutes a random walk called a Markov chain. The algorithm can be run simultaneously by several independent chains, providing an easy way to parallelize the sampling. In `PANCO2`, the sampler used is the affine-invariant sampler implemented in the `emcee` Python library [4], where the parameter space is explored by "walkers", the key difference being that walkers are not independent and communicate with each other during the sampling.

Starting point

The starting point of the Markov chains in the parameter space is an input of any MCMC analysis. The sampling can either be started from a random position – in which case the sampler needs to find the optimal region – or from an initial guess of the user. For `PANCO2`, we chose the latter, in order to speed up the process.

In gNFW mode, the parameters maximizing the posterior distribution of Eq. (3.4) are found using the `migrad` algorithm of `iMinuit`, the Python implementation of the `MINUIT` suite [2]. The MCMC is started at this position in the parameter space.

In non-parametric mode, the parameters are the pressure in each bin, and their starting point are computed as the value of the universal pressure profile of [1] at each radius bin.

For other parameters (calibration coefficient, zero level, point source fluxes), the starting point of each parameter is the maximum of its prior distribution.

Chains convergence

One crucial step of MCMC analyses is to know at which point the sampling can be stopped. To do so, the following test was implemented and is performed regularly:

1. Apply a burn-in cut, *i.e.* discard a portion of each chain at its beginning (the time needed to reach the "correct" part of the parameter space);
2. Compute the autocorrelation length l, which is the number of steps n a walker has to perform from a position θ_i so that θ_{i+n} is independent from θ_i (or, more simply put, the time it takes a walker to forget where it comes from);
3. The convergence of the chains is accepted if:
   - More than 2/3 of the chains have walked more than 40 times their autocorrelation length,
   - These chains pass the R-hat test of [5]:

$$\hat{R} = \sqrt{\frac{V}{W}} < 1.02$$

   where V measures the variance between all chains, and W measures the average variance within one chain.

The whole process ensures that more than two thirds of the chains are long enough that they can be used for inference, and that they are correctly mixed, *i.e.* that the individual properties of each chains are similar to those of the whole sample.

3.4 Results exploitation

Once the chains have reached convergence, they constitute a random sample for which the probability distribution is the posterior distribution of Eq. (3.4). These are used to infer measurements of the physical properties of the ICM.

3.4.1 The ICM pressure profile

The pressure profile of the ICM is the property directly probed by our fit. Its value is given by the computation of our model for the set of parameters that maximize the posterior distribution sampled in our fit. Each set of parameters sampled in the MCMC is then used to compute a pressure profile on a wide radius range. The dispersion of these profiles gives a measurement of the statistical error on the pressure for the whole radial range considered.

In non-parametric mode, at the end of the MCMC, another regression can be performed to fit a gNFW profile on the non-parametric pressure bins. This fit is difficult to perform, as it requires to fit a model with highly correlated parameters on a small number of data points. Sampling a posterior distribution is therefore tedious without tight priors. Still, a module is implemented to allow the user to perform such a fit, using MCMC as well as a maximum-likelihood approach is implemented, where the user may use `MIGRAD` to find the gNFW profile that best describe the pressure profile. In each case, the full posterior distribution of `PANCO2`'s first MCMC is used in the fit (*i.e.* the correlations between the different bins are taken into account).

3.4.2 Other thermodynamical profiles

If X-ray data are available for the cluster, PANCO2 will also combine them with the resulting pressure profile to infer more measurements of thermodynamical properties. Namely, the pressure profile from NIKA2 $P(r)$ will be combined with the density profile from X-ray data $n(r)$ to compute the radial profiles of the ICM temperature $k_{\rm B}T(r) = P(r)/n(r)$, entropy $K(r) = P(r)\,n^{-5/3}(r)$, hydrostatic mass through the equation of hydrostatic equilibrium (2.2). The statistical error on each profile are computed the same way as for the pressure profile (see Section 3.4.1).

3.4.3 Integrated quantities

One of the goals of the NIKA2 LPSZ is to compute the scaling relation between Y_{500} and M_{500}, namely the integrated SZ signal and mass contained in a radius R_{500}. These quantities are computed by PANCO2.

If X-ray data are available, a measurement of R_{500} can be computed by finding the radius inside which the average density contrast is 500, *i.e.* by solving

$$\delta_c(R_{500}) = \frac{M_{\rm HSE}(R_{500})}{\rho_c(z) \times \frac{4}{3}\pi R_{500}^3} = 500$$

where $\rho_c(z)$ is the critical density of the Universe at the redshift of the cluster z. Computing it for all sets of parameter sampled by our MCMC gives the probability distribution of R_{500}. Similarly, the integrated SZ signal Y_{500} and hydrostatic mass M_{500} can be computed for all the sampled pressure profiles.

Integrated quantities for non-parametric fits

As seen in Eq. (2.2), the hydrostatic mass profile is proportional to the derivative of the pressure profile with respect to radius. A consequence is that for non-parametric profiles, which are not smooth functions of the radius, mass profiles can present undesirable features such as discontinuities. This can potentially have strong effects on the measurement of R_{500} and M_{500}.

We bypass this problem by using spline interpolations. The samples of the posterior distribution (*i.e.* the value of the pressure bins sampled during the MCMC) are each interpolated in the log-log plane using a spline. The derivative of the pressure needed to evaluate the hydrostatic mass is then computed as the derivative of the spline. This smoothes the pressure profile, making its derivation more stable.

Error bars on integrated quantities

Since Y_{500} and M_{500} are both computed inside a radius R_{500}, the correlation between the three variables is obviously important. A consequence is that the values can be evaluated in two distinct ways:

- Each sampled pressure profile can be used to compute a value of R_{500}, inside which we compute the integrated quantities Y_{500} and M_{500} ;
- The best-fitting pressure profile cam be used to compute a fixed value of R_{500}, inside which we compute the integrated quantities Y_{500} and M_{500} for each sampled profile.

The latter leads to significantly smaller error bars on the integrated quantities, as it does not allow to propagate the uncertainty on R_{500} to the integrated quantities. Both ways are implemented in PANCO2, although by default the former approach will be used.

CHAPTER

FOUR

USING PANCO2

This section explains how to use PANCO2, *i.e.* how to run the code to produce results. The `git` repository includes two demonstrations that can be launched for a new user to get their hands in the code. They consist in performing a complete PANCO2 fit on a simulated map of ACT-CL J0215.4+0030, in gNFW and non-parametric mode.

To launch them, simply run:

```
$ cp Demo/demo_params_np.py panco_params.py
$ ./panco.py
```

for the non-parametric mode, or, for gNFW:

```
$ cp Demo/demo_params_gNFW.py panco_params.py
$ ./panco.py
```

4.1 The main code: `panco.py`

The main code is a compilation of the successive steps described in :numref:`PANCO2's algorithm`. `panco.py` is an executable python3 file to be called as follows:

```
$ ./panco.py [--restore=path]
```

The `--restore` argument is optional and can be passed to re-process the data of a previous PANCO2 execution, by restoring the chains rather than re-sampling. For example, if you have results in `./Results/demo` and you have modified a part of the code that does not affect the sampling, you can reprocess the results with:

```
$ ./panco.py --restore=./Results/demo/
```

`panco.py` is organized in seven sections:

1. Arguments and options

   This is where everything the user gives PANCO2 is managed, especially the `panco_params.py` configuration file – see next

2. Initializations

   This part initializes the functions that will be needed for the model evaluation, and computes everything that only needs to be computed once

3. Posterior probability distribution definition

4. MCMC starting point construction

   See Section 3.3.3

5. MCMC sampling

   Run the MCMC and monitor its progression and convergence, save chains regularly

6. Markov chains management

   Clean up the chains and format them to prepare for the inference of physical properties, perform the MCMC diagnostics

7. Results exploitation

   Convert the posterior distribution sampled during the MCMC into physical properties (thermodynamical profiles, integrated quantities) and do the plots

4.2 The `panco_params.py` configuration file

The user input part of `PANCO2` is managed via a python script called `panco_params.py`. In this script, the user defines all the options to be used in the fit, as well as the path to the input data, to the results, and many other parameters.

To run `PANCO2`, the `panco_params.py` file must be created by the user and placed in the same directory as the `panco.py` executable. When running `panco.py`, the `panco_params.py` is loaded and copied at the location where the results will be saved, so that you always have a trace of how the results were obtained when analyzing them. If the `--restore=[path]` option is given, `PANCO2` loads the options from the `panco_params.py` located in `[path]`.

The configuration files included in the demonstration include detailed comments on what options the user can personalize for both a gNFW and a non-parametric fit. Both these examples are repeated below.

/Demo/demo_params_np.py

```python
import numpy as np
from astropy.coordinates import SkyCoord
import astropy.units as u
import os

# fmt: off

### Options
do_ps      = False  # Treat point source fluxes as free parameters of the fit
fit_zl     = True   # Fit the zero level of the map
sim        = True   # Run on a simulation

model_type = "NonParam"     # "gNFW" or "Nonparam" (case insensitive)
n_bins = 4                  # number of bins between NIKA2 beam and R500
interp_method_np = "interp_powerlaw" # "interp_spline", "interp_powerlaw", or "gNFW"

crop = 6.5 * u.arcmin # Only use the NIKA2 data in a given FoV, can also be None
coords_center = SkyCoord(33.868157 * u.deg, 0.5088958 * u.deg) # Can be None

### Results paths
ana_name = "demo_nonparam"   # Name of your analysis (i.e. of the save directory)
path_to_results = f"./Results/{ana_name}/"  # Path to the results

### MCMC parameters (will be overriden if --debug option is passed)
nsteps   = 1e5  # Initial number of steps to make, can stop before if converged
ncheck   = 1e3  # Check convergence every `ncheck` steps
burn     = 1e3  # Burn-in
nchains  = 30   # Number of chains
nthreads = 30   # Number of threads to use
thin_by  = "autocorr" # Only keep one point every n steps, can be "autocorr"

### Data paths
file_nk2 = "./Demo/Data/map_input_sim.fits"
hdu_data = 4
hdu_rms  = 5
path_ps  = "./PointSources/"
file_tf  = "./Demo/Data/transfer_function.fits"
x_file   = "./Demo/Data/parprod4.json"

if sim: # Testing on simulations
    file_nk2_sim = "./Demo/Data/map_input_sim.fits"
    file_truth = "./Demo/Data/sim_truth.json"

### Cluster parameters
cluster_kwargs = {
    "z": 0.865,
    "Y_500": 1.8e-4 * u.arcmin ** 2,
    "err_Y_500": 0.5e-4 * u.arcmin ** 2,
    "theta_500": None,
}
integ_mode = "500"  # "tot" or "500", which Y is used in the likelihood

### Prior parameters, see panco_likelihood.Prior
prior_kwargs = {
    "nonparam": True,
    "calib": (-11.9, 1.19), # (Value, dispersion) for y2mJy/beam
```

(continues on next page)

(continued from previous page)

```
}

// fmt: on
```

/Demo/demo_params_gnfw.py

```python
import numpy as np
from astropy.coordinates import SkyCoord
import astropy.units as u
import os

# fmt: off

### Options
do_ps      = False  # Treat point source fluxes as free parameters of the fit
fit_zl     = True   # Fit the zero level of the map
sim        = True   # Run on a simulation

model_type = "gNFW"         # "gNFW" or "Nonparam" (case insensitive)

crop = 6.5 * u.arcmin # Only use the NIKA2 data in a given FoV, can also be None
coords_center = SkyCoord(33.868157 * u.deg, 0.5088958 * u.deg) # Can be None

### Results paths
ana_name = "demo_gNFW"   # Name of your analysis (i.e. of the save directory)
path_to_results = f"./Results/{ana_name}/"  # Path to the results

### MCMC parameters (will be overriden if --debug option is passed)
nsteps   = 1e6  # Initial number of steps to make, can stop before if converged
ncheck   = 1e4  # Check convergence every `ncheck` steps
burn     = 1e4  # Burn-in
nchains  = 30   # Number of chains
nthreads = 30   # Number of threads to use
thin_by  = "autocorr" # Only keep one point every n steps, can be "autocorr"

### Data paths
file_nk2 = "./Demo/Data/map_input_sim.fits"
hdu_data = 4
hdu_rms  = 5
path_ps  = "./PointSources/"
file_tf  = "./Demo/Data/transfer_function.fits"
x_file   = "./Demo/Data/parprod4.json"

if sim: # Testing on simulations
    file_nk2_sim = "./Demo/Data/map_input_sim.fits"
    file_truth = "./Demo/Data/sim_truth.json"

### Cluster parameters
cluster_kwargs = {
    "z": 0.865,
    "Y_500": 1.8e-4 * u.arcmin ** 2,
    "err_Y_500": 0.5e-4 * u.arcmin ** 2,
    "theta_500": None,
}
integ_mode = "500"  # "tot" or "500", which Y is used in the likelihood

### Prior parameters, see panco_likelihood.Prior
prior_kwargs = {
    "ref": "A10",           # Can also be "Planck"
    "sigma": 1.0,           # Dispersion in units of central value
    "calib": (-11.9, 1.19), # (Value, dispersion) for y2mJy/beam
    "flat": False,          # Flat priors (if False, Gaussian)
```

(continues on next page)

(continued from previous page)

```
    # "a":(2.0, 2.0),       # You can even override the slopes!
    # "b":(3.5, 1.0),
}

# fmt: on
```

4.3 Output files

When finished, PANCO2 will prompt the path where all results were saved, which is the path that has been specified as `path_to_results` in `panco_params.py`. Looking in this directory, several files can be found:

- `Plots`: a directory where your plots are saved;
- `chains.npz`: a dictionary file where the Markov chains are stored;
- `integrated_values_samples.npz`: a dictionary file the integrated values resulting from the fit, if X-ray data were available;
- `panco_params.py`: your input parameter file, for future reference;
- `radius_tab.npy`: in non parametric mode, the radial bins at which the pressure was evaluated during the fit;
- `thermo_profiles.npz`: the thermodynamical profiles computed for all points in the final PDF.

Many of these files are `.npz` files, which are files used by `numpy` to store dictionaries, similarly to `.save` files in `IDL`. Here is an example on how to read them in `python`:

```
import numpy as np
f = np.load("yourfile.npz")
print(f.files())  # Will show the available entries
a = f["a"]  # To access the data in the "a" entry
```

CHAPTER

FIVE

API

5.1 _cluster.py

class _cluster.**Cluster**(*z*, *Y_500*, *err_Y_500=None*, *R_500=None*, *theta_500=None*)

Define a galaxy cluster and its properties.

Parameters

- **z** (*float*) – redshift.
- **Y_500** (*astropy.units.Quantity*) – Integrated Compton parameter.
- **err_Y_500** (*astropy.units.Quantity*) – Optional, error on Y_500.
- **R_500** (*astropy.Units.Quantity*) – Optional, default from A10 scaling relation
- **theta_500** (*astropy.Units.Quantity*) – Optional, default from A10 scaling relation)

Notes

This class is a standalone that does not depend on any data. You can easily play around with it to check if the parameters behave as you intend.

5.2 _data.py

_data.**read_data**(*file_nk2*, *hdu_data*, *hdu_rms*, *crop=None*, *coords_center=None*, *inv_covmat=None*, *file_noise_simus=None*)

Reads input SZ data and formats it for the MCMC.

Parameters

- **file_nk2** (*str*) – path to a fits file;
- **hdu_data** (*int*) – extension containing the map to be fitted;
- **hdu_rms** (*int*) – extension containing the noise RMS map;
- **crop** (*Quantity or None*) – the size of the map to be fitted, in angle units;
- **coords_center** (*SkyCoord or None*) – the center of the map to be fitted in equatorial coordinates;
- **inv_covmat** (*array or None*) – inverse of the noise covariance matrix, in the same units as the input map to the power of -2;

- **file_noise_simus** (*str or None*) – path to a fits file where each extension is a correlated noise realization normalized to the noise RMS, to be used to compute the covariace matrix if it is not provided.

Returns

tuple containing:

- (array) the SZ map correctly cropped and centered;
- (array) the noise RMS map correctly cropped and centered;
- (array or None) the noise covariance matrix;
- (array or None) the inverse of the noise covariance matrix;
- (WCS) the world coordinate system associated to the maps;
- (Quantity) the pixel size of the maps in angle units.

Return type (tuple)

5.3 _model.py

5.3.1 The Model class

class _model.**Model**(*cluster*, *fit_zl=True*, *do_ps=False*)

Class encapsulating the tools for model computation.

Parameters **cluster** (_cluster.Cluster) – a Cluster object.

Notes

This is a parent class from which derive _model_gnfw.ModelGNFW and _model_nonparam.ModelNonParam

Model.**init_point_sources**(*path=''*, *data=None*, *wcs=None*, *reso=3*, *beam=17.6*, *ps_prior_type='pdf'*, *fixed_error=None*, *do_subtract=True*, *which_ps=None*)

Initialize everything to treat point sources. A catalog is created with the pixel positions and fluxes of the sources to fit. The ones to subtract are subtracted from the input map.

Parameters

- **path** (*str*) – path to your point sources results,
- **data** (*array*) – NIKA2 150 GHz map,
- **wcs** (*astropy.coordinates.WCS*) – WCS associated to the NIKA2 map,
- **reso** (*float*) – pixel size in arcsecs,
- **beam** (*float*) – instrumental beam FWHM in arcsec,
- **ps_prior_type** (*str*) – "pdf" or "gaussian",
- **fixed_error** (*float, None*) – If ps_prior_type = "gaussian", you can force an error bar on your prior,
- **do_subtract** (*bool*) – wether you want to remove the sources flagged as *subtract* in your catalog
- **which_ps** (*list*) – which point sources to take into account. Example: [0, 1] takes the first two sources of the catalog. If None, takes them all (super misleading)

Model.**init_transfer_function**(*tf_file*, *npix*, *pad*, *reso*)
Initialize the transfer function convolutions.

Parameters

- **tf_file** (*str*) – path to the `.fits` file with the transfer function. This recognizes files from `SZ_RDA` and `SZ_IMCM`.
- **npix** (*int*) – size of the NIKA2 map.

Model.**params_to_dict**(*params*)
Given an input parameter vector (e.g. from your MCMC), returns a dict with all the parameters called by names.

Model.**convolve_tf**(*in_map*)
Convolves a map with the transfer function, as initialized in `self.init_transfer_fnction`.

Parameters **in_map** (*array*) – The map to be convolved

Returns The convolved map

Return type (array)

Model.**__call__**(*par*)
Computes the total model map and integrated SZ signal.

Parameters **par** (*dict*) – the parameters of the model,

Returns

tuple containing:

- model_map (array) : the resulting map
- Y_500 (float) : the integrated SZ signal

Return type (tuple)

5.4 _model_gnfw.py

5.4.1 ModelGNFW

class _model_gnfw.**ModelGNFW**(*cluster*, ***kwargs*)
`_model.Model` in the case you want to perform a gNFW fit on your data.

ModelGNFW.**init_profiles_radii**(*reso=3.0*, *npix=101*, *center=0, 0*, *r_min_z=0.001*, *r_max_z=5000.0*, *nbins_z=500*, *mode='500'*, ***kwargs*)
Initialize the radii arrays necessary to compute profiles.

Parameters

- **reso** (*float*) – pixel size in arcsec,
- **npix** (*int*) – number of pixels,
- **center** (*tuple*) – pixel offset of the point at which you want your profiles to be computed (not to be confused with coords_center from `panco_params.py`)
- **r_min_z** (*floats*) – range of the line of sight considered for the analysis (tipycally 1 pc -> 5*R_500),
- **r_max_z** (*floats*) – range of the line of sight considered for the analysis (tipycally 1 pc -> 5*R_500),
- **nbins_z** (*int*) – number of bins along the line of sight.

- **mode** (*str*) – "500" or "tot", which Y to use for large scale constraints

ModelGNFW.**init_param_indices**(*nocalib=False*)
Generates the names of parameters of your model depending on the options you use. This function makes `self.params_to_dict` and `self.dict_to_params` work.

ModelGNFW.**dict_to_params**(*dic*)
Given a dict describing a parameter vector, returns a vector.

5.5 _model_nonparam.py

5.5.1 ModelNonParam

class _model_nonparam.**ModelNonParam**(*cluster*, ***kwargs*)
`_model.Model` in the case you want to perform a non-parametric fit on your data.

ModelNonParam.**init_profiles_radii**(*reso=3.0*, *npix=101*, *center=0, 0*, *mode='500'*, *radius_tab=None*, *n_bins=6*, ***kwargs*)
Initialize the radii arrays necessary to compute profiles.

Parameters

- **reso** (*float*) – pixel size in arcsec,
- **npix** (*int*) – number of pixels,
- **center** (*tuple*) – pixel offset of the point at which you want your profiles to be computed (not to be confused with coords_center from `panco_params.py`)
- **mode** (*str*) – "500" or "tot", which Y to use for large scale constraints
- **radius_tab** (*(array) or None*) – the radial binning of the profile you want to fit if you already defined it.
- **n_bins** (*int*) – if `radius_tab` is None, the number of bins to have on your pressure profile between the NIKA2 beam and R_500.

ModelNonParam.**init_param_indices**()
Generates the names of parameters of your model depending on the options you use. This function makes `self.params_to_dict` and `self.dict_to_params` work.

ModelNonParam.**dict_to_params**(*dic*)
Given a dict describing a parameter vector, returns a vector.

5.6 _probability.py

_probability.**log_lhood_nocovmat**(*par*, *model*, *data*, *error*)
Log-likelihood computation with no covariance matrix inclusion.

Parameters

- **par** (*dict*) – the model parameter values at which to evaluate the likelihood.
- **model** (*_model.Model child*) – a `Model` instance to be used to compute the model map and integrated signal.
- **data** (*array*) – your input map.
- **error** (*array*) – your 1sigma error map.

Returns the log-likelihood

Return type float

5.6.1 The Prior class

class _probability.**Prior**(*model*, *ref='A10'*, *gNFW_values=None*, *sigma=0.5*, *flat=False*, *calib=- 12.0, 1.2*, *a=None*, *b=None*, *c=None*, *nonparam=False*)

Class to compute the priors on your MCMC parameters.

Parameters

- **model** (*_model.Model child*) –
- **ref** (*None*) – name of your favorite universal pressure profile (implemented: "A10", "Planck")
- **gNFW_values** (*(array) or None*) – if you don't want to use a universal pressure profile, the central values of P0, rp, a, b, c for your priors.
- **sigma** (*float*) – Dispersion around the parameters of the gNFW, in fraction of the central value (1.0 = 100%).
- **flat** (*bool*) – wether you want to use flat priors on the gNFW parameters.
- **calib** (*tuple*) – the central value and dispersion for your y-to-Jy/beam conversion coefficient.
- **a** (*tuple or None*) – override the priors for the slopes of the gNFW.
- **b** (*tuple or None*) – override the priors for the slopes of the gNFW.
- **c** (*tuple or None*) – override the priors for the slopes of the gNFW.
- **nonparam** (*bool*) – answer to "Are you fitting a non-parametric pressure profile?"

Prior.**__call__**(*par*)

Compute the log-prior on yout model parameters.

Parameters **par** (*dict*) – the model parameter values at which to evaluate the likelihood.

Returns the log-prior

Return type float

5.7 _results.py

class _results.**Results**(*out_chains_file*, *burn*, *model*, *path*, *x_profiles=None*, *truth=None*, *dof=1*)

A class managing the results of the MCMC, and their representation.

Parameters

- **out_chains_file** (*str*) – the path to the latest save of your Markov chains.
- **burn** (*int*) – the burn-in period.
- **model** (*_model.Model child*) –
- **path** (*str*) – where to save the results and plots.
- **x_profiles** (*dict*) – a parprod-generated dict (see _xray)
- **truth** (*dict or None*) – if you ran the fit on a simulation, the true value of the parameters.

- **dof** (*int*) – the number of degrees of freedom in your statistical model to compute the reduced chi-squared.

5.7.1 Data analysis methods

`Results.`**clean_chains**(*burn_out=None*, *clip_at_sigma=None*, *clip_at_autocorr=0.0*, *thin_by=None*)

Chain management: remove a given burn-in time (and/or burn-out); if asked, clip problematic chains; if asked, isolate a subsample of points separated by the max autocorrelation length (faster for batch model computation).

Parameters

- **clip_at_sigma** (*float*) – # of sigma on the posterior distribution beyond which you want to delete chains. Can be *None* for no clipping.
- **clip_at_autocorr** (*float*) – min # of autocorrelation lengths to keep a chain. All chains that have walked shorter than that will be discarded.
- **thin_by** (*str or int*) – only keep one point every `thin_by` for each chain. Can be `'autocorr'` for the autocorrelation length of each chain.
- **burn_out** (*int*) – the opposite of burn-in, discard chains after this length.

Returns the number of chains remaining after your cuts were applied

Return type (int)

`Results.`**compute_solid_statistic**(*solid*)

Define the estimator to be used for the central value when describing a PDF.

Parameters **solid** (*str*) – which statistic to use. One of `max-like` (maximum likelihood), `max-post` (maximum posterior), or `median`.

`Results.`**chains2physics**(*radii*, *method='interp_powerlaw'*, *nthreads=8*)

Computes the physical properties from the samples of the posterior distribution and stores them in a dictionary (`self.thermo_profiles`).

Parameters

- **radii** (*array*) – the radii at which to compute the thermodynamical properties.
- **nthreads** (*int*) – the number of threads that can be used for the thermodynamical properties of the ICM,
- **method** (*str*) – how to proceed for non-parametric profiles. Can be “interp_powerlaw”, “interp_spline”, or “gNFW”

`Results.`**compute_integrated_values**(*fix_R500=False*)

Computes R_500, M_500 and Y_500 for all samples of the Markov chains to get a probability distribution for each measurement.

Parameters **fix_R500** (*bool*) – whether or not you want all your computations of M_500 and Y_500 to be performed at the same value of R_500. If False, A value of R_500 will be computed for each sample in the MCMC, and M_500 and Y_500 will be computed inside it, which will result in much larger error bars (see “Error bars on integrated quantities” in the doc).

5.7.2 Plotting methods

Results.**plot_mcmc_diagnostics**(*cleaned_chains=False*, *param_names=None*)
Plots the walks for all chains and all parameters. The plots are created using ChainConsumer, doc : https://samreay.github.io/ChainConsumer/examples/index.html

Results.**plot_distributions**(*color=None*, *param_names=None*, *alsoplot=[]*)
Plot the posterior distribution as a corner plot, as well as a marginalized distributions-only plot. The plots are created using ChainConsumer, doc : https://samreay.github.io/ChainConsumer/examples/index.html

Parameters

- **color** (*str or None*) – whether to use the color of the points as an additional dimension. Can be "likelihood" or "posterior".
- **param_names** (*list or None*) – the list of parameter names, if None, will try to figure it out using `self.model.param_names`
- **alsoplot** (*list*) – add some probability values as an additional dimension. Can contain "chi2", "lnlike", "lnprob", "lnprior".

Results.**plot_profiles**(*x=True*, *r_limits=0.1, 1.0*, *vlines={}*)
Plot the thermodynamical profiles.

Parameters

- **x** (*bool*) – whether you also want the X-ray only profiles on the plots.
- **r_limits** (*tuple*) – radial range on which to show the profiles in kpc.
- **vlines** (*dict*) – the vertical lines to add on the plot for reference. Each element should be ("String to display" : value in kpc)

Results.**plot_dmr**(*data*, *smooth_pix*, *rms=None*)
Plot data, model, residuals.

Parameters

- **data** (*array*) – the data map.
- **smooth_pix** (*float*) – the sigma of the gaussian kernel you want to apply to smooth your maps for the plot (in pixel units).

5.8 _fit_gnfw_on_non_param.py

class _fit_gnfw_on_non_param.**GNFWFitter**(*chains*, *radius_tab*, *path_to_results*, *cluster*, *kind='gaussian'*, *which_pts=slice(None, None, None)*, *center_bins=None*)
Class to fit a gNFW model from non-parametric results.

Parameters

- **chains** (*array*) – the chains from your non-parametric MCMC, shape = (nwalkers, nsteps, nparams)
- **radius_tab** (*array*) – the radial bins of your non-parametric profile
- **path_to_results** (*string*) – where to save your results,
- **cluster** (_cluster.Cluster) – a `Cluster` object,

- **kind** (*string*) – "gaussian" or "kde", how you want the probability distribution of the pressure points to be computed.
- **which_pts** (*slice*) – which of the nonparam points to be fitted. May be used to e.g. remove the last two points, with slice(0, -2)
- **center_bins** (*None or (array)*) – override the center of the marginalized distributions for the pressure bins. May be used to e.g. center on best-fit.

GNFWFitter.**do_fit_minuit**()
Perform the fit using `Minuit`, i.e. find the maximum-likelihood for a gNFW profile on your non-parametric posterior distribution

GNFWFitter.**do_fit_mcmc**(*nchains*, *nsteps*, *ncheck*, *nthreads*, *burn*, *restore=False*)
Perform the fit using MCMC, i.e. sample a posterior distribution for a gNFW profile on your non-parametric posterior distribution

Parameters

- **nchains** (*int*) – the number of walkers.
- **nsteps** (*int*) – the initial number of steps to perform (sampling can stop before that if convergence has been reached).
- **ncheck** (*int*) – MCMC convergence will be checked every `ncheck` iterations.
- **nthreads** (*int*) – number of threads to use.
- **burn** (*int*) – the burn-in period.
- **restore** (*bool*) – wether you want to read existing chains rather than sampling.

GNFWFitter.**manage_chains**(*burn*)
Performs chain cleaning and MCMC diagnostic plots

Parameters **burn** (*int*) – the burn-in period.

GNFWFitter.**compute_thermo_profiles**(*radii*, *x_profiles=None*, *nthreads=100*)
Computes the thermodynamical profiles from the sampled posterior distribution

Parameters

- **radii** (*array*) – the radii at which to compute the thermodynamical properties.
- **x_profiles** (*dict*) – a parprod-generated dict (see `_xray`)
- **nthreads** (*int*) – the number of threads that can be used for the thermodynamical properties of the ICM.

5.9 _xray.py

_xray.**recover_x_profiles**(*x_file*)
Open the X-ray parprod and returns the profiles.

Parameters **x_file** (*str*) – the parprod file. Can be in .fits or in .json; if you only have a .save, use */data/Workspace/SZ_program_data/XMM_files/parprod2json.pro*

Returns

a dict with the thermodynamical profiles of the parprod. Keys are:

- `rd`, `d`, `errd`: radius [kpc], density and error [cm-3],
- `rt`, `t`, `errt`: radius [kpc], temperature and error [keV],

- rp, p, errp: radius [kpc], pressure and error [keV cm-3],
- rk, k, errk: radius [kpc], entropy and error [keV cm2],
- rm, m, errm: radius [kpc], HS mass and error [Msun]

Return type (dict)

5.10 _utils.py

class _utils.**LogLogSpline**(*x*, *y*, ***kwargs*)

Performs a spline interpolation in the log-log plane. The interface is the same as *scipy.interpolate.UnivariateSpline*, but the interpolation is performed on the log10 of your data.

Parameters

- **x** (*array*) – input *x*-axis.
- **y** (*array*) – input *y*-axis
- ****kwargs** – kwargs to pass scipy.interpolate.UnivariateSpline

LogLogSpline.**__call__**(*x*)

Computes the interpolation at a given *x*.

Parameters **x** (*array of float*) – the *x* value(s) at which to perform the interpolation

Returns (array or float) *f(x)*

LogLogSpline.**differentiate**(*x*, *n=1*)

Computes the *n*-th derivative of the spline in *x*.

Parameters

- **x** (*array of float*) – the *x* value(s) at which to perform the interpolation
- **n** (*int*) – degree of differentiation.

Returns (array or float) d`f`/d`x` *(x)*

_utils.**interp_powerlaw**(*x*, *y*, *x_new*, *axis=0*)

Interpolate/extrapolate a profile with a power-law by performing linear inter/extrapolation in the log-log space.

Parameters

- **x** (*array*) – input x-axis.
- **y** (*array*) – input *f(x)*.
- **x_new** (*float or array*) – *x* value(s) at which to perform interpolation.

Returns *f(x_new)*

Return type (float or array)

_utils.**adim**(*qty*)

Removes the dimension of an adimensioned astropy quantity, e.g. >>> adim(2.0 * (u.m / u.cm)) 200.0

Parameters **qty** (*astropy.units.Quantity*) – a dimensioned quantity.

Returns the adimensioned quantity.

Return type (float)

_utils.**sph_integ_within**(*prof*, *r*, *axis=0*)

Integration of a profile in spherical coordinates between min(r) and max(r)

Parameters

- **prof** (*array*) – the profile to integrate along the x axis.
- **r** (*array*) – corresponding radii vector.

Returns $4\pi \int_{min(r)}^{max(r)} r^2 prof(r) dr$

Return type (float)

_utils.**cyl_integ_within**(*prof*, *r*, *axis=0*)

Integration of a profile in cylindrical coordinates between min(r) and max(r)

Parameters

- **prof** (*array*) – the profile to integrate along the x axis.
- **r** (*array*) – corresponding radii vector.

Returns $2\pi \int_{0}^{max(r)} r prof(r) dr$

Return type (float)

_utils.**prof2map**(*prof_x*, *r_x*, *r_xy*)

Compute a 2D map from a 1D profile through power law interpolation.

Parameters

- **prof_x** (*array*) – the profile along the *x* axis.
- **r_x** (*array*) – corresponding radii vector.
- **array)** (*r_xy*) – radii map in the *(x, y)* plane.

Returns Profile interpolated in the (x, y) plane

Return type (array)

BIBLIOGRAPHY

[1] M. Arnaud, G. W. Pratt, R. Piffaretti, H. Böhringer, J. H. Croston, and E. Pointecouteau. The universal galaxy cluster pressure profile from a representative sample of nearby systems (REXCESS) and the YSZ – M500 relation. *Astronomy & Astrophysics*, 517:A92, July 2010. URL: https://www.aanda.org/articles/aa/abs/2010/09/aa13416-09/aa13416-09.html, doi:10.1051/0004-6361/200913416.

[2] Hans Dembinski, Piti Ongmongkolkul, Christoph Deil, David Menéndez Hurtado, Matthew Feickert, Henry Schreiner, Andrew, Chris Burr, Fabian Rost, Alex Pearce, Lukas Geiger, Bernhard M. Wiedemann, and Omar Zapata. Scikit-hep/iminuit: v1.4.9. July 2020. URL: https://zenodo.org/record/3951328, doi:10.5281/zenodo.3951328.

[3] Planck Collaboration et al. Planck intermediate results - V. Pressure profiles of galaxy clusters from the Sunyaev-Zeldovich effect. *Astronomy & Astrophysics*, 550:A131, February 2013. URL: https://www.aanda.org/articles/aa/abs/2013/02/aa20040-12/aa20040-12.html, doi:10.1051/0004-6361/201220040.

[4] Daniel Foreman-Mackey, Will M. Farr, Manodeep Sinha, Anne M. Archibald, David W. Hogg, Jeremy S. Sanders, Joe Zuntz, Peter K. G. Williams, Andrew R. J. Nelson, Miguel de Val-Borro, Tobias Erhardt, Ilya Pashchenko, and Oriol Abril Pla. Emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC. *arXiv e-prints*, pages arXiv:1911.07688, November 2019. URL: https://ui.adsabs.harvard.edu/abs/2019arXiv191107688F/abstract.

[5] Andrew Gelman and Donald B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7:457–472, 1992. URL: http://adsabs.harvard.edu/abs/1992StaSc...7..457G, doi:10.1214/ss/1177011136.

[6] F. Kéruzoré, F. Mayet, G. W. Pratt, R. Adam, P. Ade, P. André, A. Andrianasolo, M. Arnaud, H. Aussel, I. Bartalucci, A. Beelen, A. Benoît, S. Berta, O. Bourrion, M. Calvo, A. Catalano, M. De Petris, F.-X. Désert, S. Doyle, E. F. C. Driessen, A. Gomez, J. Goupy, C. Kramer, B. Ladjelate, G. Lagache, S. Leclercq, J.-F. Lestrade, J. F. Macías-Pérez, P. Mauskopf, A. Monfardini, L. Perotto, G. Pisano, E. Pointecouteau, N. Ponthieu, V. Revéret, A. Ritacco, C. Romero, H. Roussel, F. Ruppin, K. Schuster, S. Shu, A. Sievers, and C. Tucker. Exploiting NIKA2/XMM-Newton imaging synergy for intermediate mass, high-\$z\$ galaxy clusters within the NIKA2 SZ Large Program. *arXiv e-prints*, 2009:arXiv:2009.02563, September 2020. URL: http://adsabs.harvard.edu/abs/2020arXiv200902563K.

[7] F. Mayet, R. Adam, P. Ade, P. André, A. Andrianasolo, M. Arnaud, H. Aussel, I. Bartalucci, A. Beelen, A. Benoît, A. Bideaud, O. Bourrion, M. Calvo, A. Catalano, B. Comis, M. De Petris, F.-X. Désert, S. Doyle, E. F. C. Driessen, A. Gomez, J. Goupy, F. Kéruzoré, C. Kramer, B. Ladjelate, G. Lagache, S. Leclercq, J.-F. Lestrade, J. F. Macías-Pérez, P. Mauskopf, A. Monfardini, L. Perotto, G. Pisano, E. Pointecouteau, N. Ponthieu, G. W. Pratt, V. Revéret, A. Ritacco, C. Romero, H. Roussel, F. Ruppin, K. Schuster, S. Shu, A. Sievers, C. Tucker, and R. Zylka. Cluster cosmology with the NIKA2 SZ Large Program. *EPJ Web of Conferences*, 228:00017, 2020. Publisher: EDP Sciences. URL: https://www.epj-conferences.org/articles/epjconf/abs/2020/04/epjconf_mmuniverse2019_00017/epjconf_mmuniverse2019_00017.html, doi:10.1051/epjconf/202022800017.

[8] Daisuke Nagai, Andrey V. Kravtsov, and Alexey Vikhlinin. Effects of Galaxy Formation on Thermodynamics of the Intracluster Medium. *The Astrophysical Journal*, 668(1):1–14, October 2007. URL: https://doi.org/10.1086%2F521328, doi:10.1086/521328.

[9] C. Romero, M. McWilliam, J.-F. Macías-Pérez, R. Adam, P. Ade, P. André, H. Aussel, A. Beelen, A. Benoît, A. Bideaud, N. Billot, O. Bourrion, M. Calvo, A. Catalano, G. Coiffard, B. Comis, M. de Petris, F.-X. Désert, S. Doyle, J. Goupy, C. Kramer, G. Lagache, S. Leclercq, J.-F. Lestrade, P. Mauskopf, F. Mayet, A. Monfardini, E. Pascale, L. Perotto, G. Pisano, N. Ponthieu, V. Revéret, A. Ritacco, H. Roussel, F. Ruppin, K. Schuster, A. Sievers, S. Triqueneaux, C. Tucker, and R. Zylka. A multi-instrument non-parametric reconstruction of the electron pressure profile in the galaxy cluster CLJ1226.9+3332. *Astronomy & Astrophysics*, 612:A39, April 2018. URL: https:/articles/aa/abs/2018/04/aa31599-17/aa31599-17.html, doi:10.1051/0004-6361/201731599.

[10] Hongsheng Zhao. Analytical models for galactic nuclei. *Monthly Notices of the Royal Astronomical Society*, 278:488–496, January 1996. URL: http://adsabs.harvard.edu/abs/1996MNRAS.278..488Z, doi:10.1093/mnras/278.2.488.

[11] F. Kéruzoré, F. Mayet, L. Perotto, and J.-F. Macías-Pérez. PSTools: A toolbox for the estimation of the contamination by sub-millimeter galaxies in NIKA2 galaxy cluster observations. 2019.

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

G

I

L

M

P

R

S