furspy

Shikhar Mittal

Mar 10, 2024

CONTENTS:

CHAPTER

ONE

BASICS

1.1 Overview

Name
Foregrounds due to Unresolved Radio Sources

Author
Shikhar Mittal

Homepage
https://github.com/shikharmittal04/furs

1.2 Why do you need this code?

Use this code to generate the Foregrounds due to Unresolved Radio Sources (FURS).

A cosmological global 21-cm signal hides under foregrounds due to galactic and extragalactic emissions. These foregrounds can easily be 4 to 5 orders of magnitude higher than the signal of interest. For a reliable inference it is important to accurately model these foregrounds. While we have a reasonable understanding of galactic emission (typically fit as log-log polynomial), we do not understand the extragalactic contributions. Based on existing models, this code models the foregrounds due to unresolved extragalactic radio sources.

Read more about it in the paper Mittal et al (2024).

1.3 Installation and requirements

This package can be installed as

```
pip install furspy
```

It is recommended to work on a Python version > 3.8. Packages required are

- numpy
- scipy
- matplotlib
- mpi4py
- healpy
- transformcl

1.4 Quick start

The code is run in two main steps:

- Assign the unresolved sources flux densities (at a chosen reference frequency) and spectral indices.
- Then generate the sky maps at desired frequencies of observation.

The following code captures the main functionalities of this package.

```
from furspy import furs

#Step-1 initialise the object with default settings
obj = furs.furs()

#Step-2 generate the data at the reference frequency
obj.ref_freq()

#Step-3 generate the sky maps at multiple frequencies as well as their sky average
obj.gen_freq()

#Step-4 finally, generate a sky averaged spectrum vs frequency figure
obj.visual()
```

Save the above code as (say) `eg_script.py` and run it as

```
python eg_script.py
```

Running the code will generate several files. The terminal messages will guide you to these output files. The most important of all files of your interest will be `Tb_nu_map.npy`. However, you may never have to deal with them yourself. To visualise your outputs use the function `visual()`. See *API Reference* for the available features for `visual()`.

The default values have been chosen such that the above script can be run on a PC. Since modern PCs have at least 4 cores, for a better performance one could also run the code as

```
mpirun -np 4 python eg_script.py
```

However, in general and for more realistic flux density ranges and high resolution maps, it is recommended to run the code on HPCs.

1.5 License and citation

The software is free to use on the MIT open source license. If you use the software then please consider citing Mittal et al (2024).

CHAPTER

TWO

DETAILED EXPLANATION

2.1 Initialisation

The first step to use this package is to initialise the properties of the unresolved sources. This is done using the class *furs.furs*. If you give no arguments default settings are assumed.

There are total 11 available optional arguments. See the *API Reference*. If you want to chose a different set of parameters, then your python script should have the following initialisation

```
from furspy import furs
obj = furs.furs(log2Nside=6, logSmin=-2,logSmax=-1,dndS_form=0, nu_o=150e6, beta_o=2.681,
↪sigma_beta=0.5, amp=7.8e-3,gam=0.821, path='', lbl='')
```

(Replace the above values by values of your choice.)

2.2 Reference frequency

The function `ref_freq()` does 3 tasks:-

1. Calcuates the total number of unresolved sources corresponding to your specified `logSmin` and `logSmin`. Then it creates a 'clustered' sky density of unresolved radio sources, fluctuation for which follows the 2PACF whose parameters are set by `amp` and `gam`.
2. Next, it visits each pixel on the sky and assigns each source a flux density chosen from a flux distribution function, $\mathrm{d}n/\mathrm{d}S$
3. Finally, it assigns a spectral index to each source which is normally distributed. The normal distribution is set by `beta_o` and `sigma_beta`.

Sky pixelisation is set by `log2Nside`. The number of pixels is $N_{\rm pix} = 12 \times 2^{2k}$, where $k =$ `log2Nside`.

The function does not return anything, but produces 4 output files, namely `n_clus.npy`, `Tb_o_individual.npy`, `Tb_o_map.npy`, and `beta.npy` in the path specified by `path` during initialisation. The files are described below.

- `n_clus.npy` is a 1D array which stores number density of unresolved radio sources as number per pixel. `n_clus[i]` gives the number of sources on the $i^{\rm th}$ pixel, where $i = 0, 1, \ldots, N_{\rm pix} - 1$. Note that in general `n_clus[i]` will not be a natural number; we simulate for a rounded-off value.
- Both `Tb_o_individual.npy` and `beta.npy` are array of arrays of unequal sizes and share equal amount of memory. Typically, these files will be huge (for default settings they will of size ~ 17 MB each). Each of `Tb_o_individual[0]`, `Tb_o_individual[1]`, ..., is an array and they are total $N_{\rm pix}$ in number corresponding to $N_{\rm pix}$ pixels. The last array is `Tb_o_individual[Npix-1]`. The length of array `Tb_o_individual[i]` is equal to the number of sources on the $i^{\rm th}$ pixel, which is `round(n_clus[i])`. The values itself are the brightness temperature contributed by each source in kelvin at reference frequency. Note that `Tb_o_individual.npy` and

`beta.npy` are 'Object' arrays. If you want to load them yourself then set `allow_pickle=True` in `numpy.load()`

- The structure of `beta` is same as `Tb_o_individual`. The values itself are the spectral indices assigned to each source.
- `Tb_o_map` is an array similar in structure to `n_clus`. It is the pixel wise brightness temperature contributed by the extragalactic radio sources at the reference frequency. Thus, `Tb_o_map[i] = numpy.sum(Tb_o_individual[i])`.

The following should be your python script

```
from furspy import furs

obj = furs.furs()

obj.ref_freq()
```

2.3 General frequency

The next important task is performed by the function `gen_freq()`. It scales the brightness temperature at reference frequency for each source according to a power law to a desired range of frequencies. The desired frequencies should be supplied (in Hz) as a `numpy` array to this function. For example the following should be your python script

```
from furspy import furs

obj = furs.furs()

obj.ref_freq()

obj.gen_freq(nu = 1e6*numpy.arange(50,201))
```

The default value of frequencies at which `gen_freq()` will scale is $\nu = 50, 51, \ldots, 200$ MHz. This function does not return anything but produces 3 files namely `Tb_nu_map.npy`, `Tb_nu_glob.npy`, and `nu_glob.npy` in the path specified by `path` during initialisation. The files are described below.

1. `Tb_nu_map` is a 2D array of shape $N_{\rm pix} \times N_\nu$, so that `Tb_nu_map[i,k]` gives the brightness temperature on the $i^{\rm th}$ pixel at `nu[k]` frequency. N_ν is the number of frequencies you gave in the argument of `gen_freq()`.
2. `Tb_nu_glob` is derived directly from `Tb_nu_map`. It is the sky average of the map at each frequency and is thus a 1D array. It is calculated as `Tb_nu_glob = numpy.mean(Tb_nu_map,axis=0)`.
3. `nu_glob.npy` is simply the frequency array you gave else it is the default value.

Note that this function loads `Tb_o_individual.npy` and `beta.npy`. These files can easily be 10s of GB in size for 'realistic' `logSmin` and `logSmax`. Common personal computers have ~ 4 GB RAM. It is thus recommended to run this code on supercomputers. For job submission scipt users are requested to specify `#SBATCH --mem-per-cpu=[size in MB]`, where a recommendation for `size in MB` will be printed by `ref_freq()` function.

2.4 Chromatic distortions

`Tb_nu_map` and hence `Tb_nu_glob` so generated do NOT account for chromatic distortions. They are simply the model outputs for foregrounds due to unresolved radio sources. However, in reality because of the chromatic nature of the antenna beam the actual foregrounds spectrum registered will be different. You can use the function `couple2D()` to account for the chromaticity. It essentially couples the foregrounds to the beam directivity.

Since this is experiment specific you will need to provide an external data file: the beam directivity pattern, D. This should be a 2D array of shape $N_{\rm pix} \times N_\nu$, such that `D[i,k]` should give the beam directivity at $i^{\rm th}$ pixel at `nu[k]` frequency. The frequencies at which you generate your data D should be the same as the frequencies you gave in `gen_freq()`. (In case you forgot, `gen_freq()` will have saved the frequency array in your `obj.path` path.) Put this array D in your `obj.path` path by the name of `D.npy`.

Only after running `ref_freq()` and `gen_freq()`, run `couple2D()` as

```
from furspy import furs

obj = furs.furs()

obj.ref_freq()

obj.gen_freq()

#If you have already ran ref_freq and gen_freq previously then comment
#obj.ref_freq() and obj.gen_freq().
obj.couple2D()
```

No input argument is required. The return value is `None`. This function will generate a file called `T_ant.npy` in your path. This will be a 1D array with length of number of frequencies.

This function will also print the best-fitting parameters (along with 1σ uncertainty) $T_{\rm f}, \beta_{\rm f}$ and $\Delta\beta_{\rm f}$ based on a simple least-squares fitting of power-law-with-a-running-spectral-index function as follows

$$T_{\rm f} \left(\frac{\nu}{\nu_0}\right)^{(-\beta_{\rm f} + \Delta\beta_{\rm f}\ \ln \nu/\nu_0)}$$

to the antenna temperature data.

2.5 Visualisation

The final part of the code is to visualise the results. Main data for inspection is in the file `Tb_nu_map.npy`. Each of `Tb_nu_map[:,k]` is an array in the standard ring ordered `HEALPix` format and is thus ready for visualisation as a Mollweide projection. You may also be interested in inspecting the global spectrum of extragalactic emission, i.e, temperature as a function of frequency. This is simply the data in the file `Tb_nu_glob.npy` generated by `gen_freq()`.

You may use the function `visual()` for both the above purposes. It is possible to make several other additional figures by simply setting the optional arguments to `True` (see below). This function is again a method of class object *furs.furs* and is thus your python script should contain

```
from furspy import furs

obj = furs.furs()

obj.ref_freq()
```

(continues on next page)

(continued from previous page)

```
obj.gen_freq()

obj.couple2D()

#comment out obj.ref_freq(), obj.gen_freq(), obj.couple2D() if you have already run them.
obj.visual()
```

For all the available options for this function see the *API Reference*. This function will produce figures in the path specficied during initialisation.

CHAPTER

THREE

OTHER FUNCTIONS

The main functionality of the package was already discussed in previous sections. Here we will learn about 5 additional useful methods of the class *furs.furs*. These are

1. The 2-point angular correlation function (2PACF, $C = C(\chi)$), `acf()`
2. The flux density distribution function ($\mathrm{d}n/\mathrm{d}S$), `dndS()`. As this function is a method of class *furs.furs*, the choice of the form of of $\mathrm{d}n/\mathrm{d}S$ will have been set when you initialise your class *furs.furs* object as

```
from furpy import furs

obj = furs.furs()
```

3. Total number of sources on the full sky (N_s), `num_sources()`. This function has direct correspondence with the form of $\mathrm{d}n/\mathrm{d}S$.
4. Number of source per pixel (n_cl), `num_den()`. As in the `num_sources()`, the total number density returned will correspond to the chosen form of $\mathrm{d}n/\mathrm{d}S$.
5. `print_input()` can be useful if you want see what parameters you are currently running with.

Note that these functions are computationally cheap and thus can be run directly in a jupyter notebook or interactively on the terminal.

Examples

Suppose you want to find the number of sources between $S = 10^{-6}$ and 10^{-1}Jy for a 7^{th} order log-log polynomial fit to Euclidean number count then do

```
>>> from furspy import furs
>>> obj = furs.furs(dndS_form=1, logSmin=-6,logSmax=-1)
>>> Ns = obj.num_sources()
>>> Ns
249825819.67727068
```

Let us calculate the number density.

```
>>> ncl = obj.num_den()
Total number of sources, Ns = 249825820
Total number of pixels, Npix = 49152
Average number of sources per pixel, n_bar = 5082.72
Done.
Average overdensity for the clustered sky (should be ~ 0) = 0.007
The clustered number density has been saved into file:
 n_clus
```

(Note that since this is a random process you might get different numbers.)

Let us also check if the sum of the elements of array `ncl` is `Ns`

```
>>> ncl.sum()
251605169.7207427
```

There some discrepancy but do not be worried as this is just a numerical artefact. In fact the fractional error is already printed and is about 0.7‰, which is sufficiently small.

CHAPTER

FOUR

GENERAL REMARKS

Users do not have to run `ref_freq()` everytime. If they want to use the same data for source distribution (`n_clus.npy`), flux density (`Tb_o_individual.npy`) and spectral index (`beta.npy`) assignments at reference frequency to generate spectrum and sky maps for a different frequency range, then run only `gen_freq()` for a new choice of `nu`.

Similarly, if you have already run `gen_freq()` and are happy with the specifications of the model then you can directly jump to the `visual()` function.

In case you forgot what data set you generated with what parameter specifications, you can always save your class object using the function `save_furs()` in the directory where all other outputs are saved and load back using `load_furs()`. (Both functions are part of module `furs.py`.)

Thus, after initialising your class object (i.e. `obj = furs.furs([YOUR SPECIFICATIONS])`), you can add to your script `furs.save_furs(obj,'myobj')`.

Examples

```
from furspy import furs

obj = furs.furs()
furs.save_furs(obj,'myobj')
```

So when you came back next time you can load it as

```
from furspy import furs
obj=furs.load_furs('/give/full/path/to/myobj.pkl')
```

Remember to give the full path to the `myobj` with the extension `.pkl`.

You may now check that indeed the specfications are correctly loaded by printing them using function `print_input()`.

```
from furspy import furs
obj=furs.load_furs('/give/full/path/to/myobj.pkl')
obj.print_input()
```

CHAPTER

FIVE

API REFERENCE

This is the module *furs*. It contains public functions *save_furs()*, *load_furs()* and a class *furs*.

furs.**save_furs**(*obj*, *filename*)

Saves the class object *furs*.

Save the class object *furs* for later use. It will save the object in the path where you have all the other outputs form this package.

5.1 Parameters

obj
: [class] This should be the class object you want to save.

filename
: [str] Give a filename to your object. It will be saved in the `obj.path` directory.

furs.**load_furs**(*filename*)

To load the class object *furs*.

5.2 Parameters

filename
: [str] This should be the name of the file you gave in *save_furs()* for saving class object *furs*. Important: provide the full path for `filename` with the extension `.pkl`.

5.3 Returns

class object

class furs.**furs**(*beta_o=2.681*, *sigma_beta=0.5*, *logSmin=-2*, *logSmax=-1*, *dndS_form=0*, *log2Nside=6*, *nu_o=150000000.0*, *amp=0.0078*, *gam=0.821*, *path=''*, *lbl=''*)

This is class for initialising the properties of the unresolved radio sources.

5.4 Attributes

nu_o
[float, optional] Reference frequency in Hz

beta_o
[float, optional] Mean spectral index for extragalactic point sources

sigma_beta
[float, optional] Spread in the beta values

amp
[float, optional] Amplitude of the power-law 2-point angular correlation function (2PACF)

gam
[float, optional] $-$ exponent of the power-law 2-point angular correlation function

logSmin
[float, optional] $\log_{10}(S_{\rm min})$, where $S_{\rm min}$ is in Jy

logSmax
[float, optional] $\log_{10}(S_{\rm max})$, where $S_{\rm max}$ is in Jy

dndS_form
[int, optional] Choose the functional form for $\mathrm{d}n/\mathrm{d}S$. Available options -> 0 (default),1 or 2

log2Nside
[int, optional] Number of divisions in units of $\log_2$

path
[str, optional] Path where you would like to save and load from, the Tb's and beta's

lbl
[str, optional] Append an extra string to all the output files.

5.5 Methods

acf(*chi*)

2 point angular correlation function.

This is the popular form of the 2PACF; a power law.

The default values for amplitude and index are from Rana & Bagla (2019).

5.5.1 Parameters

chi
[float] Angle at which you want to get the 2PACF, should be in radians. One number or an array.

5.5.2 Returns

float
Output is pure number or an array accordingly as `chi` is a number or an array.

couple2D()

Couple the foregrounds generated by *gen_freq()* to the antenna beam directivity.

`Tb_nu_map.npy` generated by *gen_freq()* does not account for chromatic distortions in the antenna. To account for this users must provide an array, named `D.npy`, which should be in the shape of $N_{\rm pix} \times N_\nu$. Put this array into the path where you have all the other outputs. An output file will be generated called `T_ant.npy`.

Also, the best-fitting parameters (along with 1σ uncertainty) $T_{\rm f}$, $\beta_{\rm f}$ and $\Delta\beta_{\rm f}$ based on a simple least-squares fitting of power-law-with-a-running-spectral-index function to the antenna temperature data will be printed.

dndS(S)

$\mathrm{d}n/\mathrm{d}S\,(\mathrm{sr}^{-1}\mathrm{Jy}^{-1})$

Distribution of flux density, S.

The default choice (0) is by Gervasi et al (2008). It is approximately a power law. See paper for the exact form.

Form 1 is by Mandal et al. (2021). It is a $7^{\rm th}$ order log-log polynomial fit to the Euclidean number count.

Form 2 is by Intema et al. (2017). It is a $5^{\rm th}$ order log-log polynomial fit to the Euclidean number count.

5.5.3 Parameters

S
[float] Flux density in units of Jy (jansky). Can be 1 value or an numpy array.

5.5.4 Returns

float
Number of sources per unit solid angle per unit flux density. 1 value or an array depending on input.

gen_freq(*nu=array([5.00e+07, 5.10e+07, 5.20e+07, 5.30e+07, 5.40e+07, 5.50e+07, 5.60e+07, 5.70e+07, 5.80e+07, 5.90e+07, 6.00e+07, 6.10e+07, 6.20e+07, 6.30e+07, 6.40e+07, 6.50e+07, 6.60e+07, 6.70e+07, 6.80e+07, 6.90e+07, 7.00e+07, 7.10e+07, 7.20e+07, 7.30e+07, 7.40e+07, 7.50e+07, 7.60e+07, 7.70e+07, 7.80e+07, 7.90e+07, 8.00e+07, 8.10e+07, 8.20e+07, 8.30e+07, 8.40e+07, 8.50e+07, 8.60e+07, 8.70e+07, 8.80e+07, 8.90e+07, 9.00e+07, 9.10e+07, 9.20e+07, 9.30e+07, 9.40e+07, 9.50e+07, 9.60e+07, 9.70e+07, 9.80e+07, 9.90e+07, 1.00e+08, 1.01e+08, 1.02e+08, 1.03e+08, 1.04e+08, 1.05e+08, 1.06e+08, 1.07e+08, 1.08e+08, 1.09e+08, 1.10e+08, 1.11e+08, 1.12e+08, 1.13e+08, 1.14e+08, 1.15e+08, 1.16e+08, 1.17e+08, 1.18e+08, 1.19e+08, 1.20e+08, 1.21e+08, 1.22e+08, 1.23e+08, 1.24e+08, 1.25e+08, 1.26e+08, 1.27e+08, 1.28e+08, 1.29e+08, 1.30e+08, 1.31e+08, 1.32e+08, 1.33e+08, 1.34e+08, 1.35e+08, 1.36e+08, 1.37e+08, 1.38e+08, 1.39e+08, 1.40e+08, 1.41e+08, 1.42e+08, 1.43e+08, 1.44e+08, 1.45e+08, 1.46e+08, 1.47e+08, 1.48e+08, 1.49e+08, 1.50e+08, 1.51e+08, 1.52e+08, 1.53e+08, 1.54e+08, 1.55e+08, 1.56e+08, 1.57e+08, 1.58e+08, 1.59e+08, 1.60e+08, 1.61e+08, 1.62e+08, 1.63e+08, 1.64e+08, 1.65e+08, 1.66e+08, 1.67e+08, 1.68e+08, 1.69e+08, 1.70e+08, 1.71e+08, 1.72e+08, 1.73e+08, 1.74e+08, 1.75e+08, 1.76e+08, 1.77e+08, 1.78e+08, 1.79e+08, 1.80e+08, 1.81e+08, 1.82e+08, 1.83e+08, 1.84e+08, 1.85e+08, 1.86e+08, 1.87e+08, 1.88e+08, 1.89e+08, 1.90e+08, 1.91e+08, 1.92e+08, 1.93e+08, 1.94e+08, 1.95e+08, 1.96e+08, 1.97e+08, 1.98e+08, 1.99e+08, 2.00e+08])*)

Scale the brightness temperature at reference frequency to a general frequency.

If you are running this function you must have run *ref_freq()*.

This function computes the map(s) at general frequency(ies) based on the precomputed values from *ref_freq()*.

5.5.5 Parameters

nu
: [float] frequency (in Hz) at which you want to evaluate the brightness temperature map. Can be one number or an array. (Default = *1e6*np.arange(50,201)*)

3 files will be generated namely, `Tb_nu_glob.npy`, `Tb_nu_glob.npy`, and `nu_glob.npy`.

To understand the structure of these output files see the section on *General frequency*.

num_den()
: Number density.

This function calculates the number density function $n_{\rm cl}$ for the 2PACF defined in `acf()`.

The array will also be saved as an `.npy` format file in the path you gave during initialisation.

5.5.6 Returns

float
: Number of sources per pixel. It will be an array of length $N_{\rm pix}$.

num_sources()
: This function gives the total number of unresolved point sources on the full sky.

This is specifically for the flux density distribution defined in *dndS()*, and the minimum and maximum S values are set during the initialisation of the class object *furs*.

5.5.7 Returns

The total number of unresolved point sources. It is a pure number.

print_input()
: Print the input parameters you gave.

ref_freq()
: Generates the brightness temperature and spectral indices at reference frequency.

3 output files are generated `Tb_o_individual.npy`, `Tb_o_map.npy` and `beta.npy`

To understand the structure of these output files see the section on *Reference frequency*.

visual(*t_skymap=False*, *nu_skymap=None*, *aps=False*, *n_skymap=False*, *dndS_plot=False*, *spectrum=True*, *antenna=False*, *xlog=False*, *ylog=True*, *fig_ext='pdf'*)
: Plotting function.

This function can produce several figures such as:-

- number density map
- FURS map
- angular power spectrum
- flux density distribution function
- sky averaged FURS as function of frequency
- antenna temperature

5.5.8 Parameters

t_skymap
[bool, optional] Want to plot the FURS map (a Mollweide projection plot)? (Default = `False`).

nu_skymap
[float, optional] Frequency in Hz at which you want to construct the FURS map. Relevant only when you give `t_skymap = True`. (Default = `nu_o`)

aps
[bool, optional] Want to plot the angular power spectrum? (Default = `False`)

n_skymap
[bool, optional] Want to plot the number density map (a Mollweide projection plot)? (Default = `False`).

dndS
[bool, optional] Want to plot the flux density distribution? (Default = `False`). The form of $\mathrm{d}n/\mathrm{d}S$ is set during initialisation.

spectrum
[bool, optional] Want to plot the sky averaged FURS? (Default = `True`).

antenna
[bool, optional] Add the antenna temperature? (Default = `False`). You should have run *couple2D()* to use this.

xlog
[bool, optional] Set the x-axis scale of spectrum plot in log? (Default = `False`)

ylog
[bool, optional] Set the y-axis scale of spectrum plot in log? (Default = `False`)

fig_ext
[str, optional] What should be the format of the figure files? Common choices include png, pdf or jpg. (Default = `pdf`)

PYTHON MODULE INDEX

INDEX