

# The Sun's Atmosphere: Microwave data analysis project

Taotao Yang

*School of Physics & Astronomy, University of Glasgow*

(Dated: April 02, 2023; Edit: April 11, 2023)

The project focus on the data analysis of the source data obtained from Nobeyama Radio Polarimeters (NoRP) and Radio Solar Telescope Network (RSTN). The project use python to read, organize, process, fit, and plot the radio data. The investigated data is centered around a flare event on October 28th, 2013, peaking at 01:59:38 UTC, with a frequency range of 0.245 GHz to 80 GHz. The project produce curve fits on Gyrosynchrotron emissions by adjusting the source data with a fitted plasma emission model. The Gyrosynchrotron fit after plasma model correction shows a better curve fit result than the original fit, where the plasma emissions at lower frequencies are included.

## I. Introduction

The sun is the major driving force of space weather, and the study of flares and their associated activities could provide us with important insights on the generation mechanics, the profiles, and influences of these electromagnetic emissions coming from the solar atmosphere [1]. The effect of solar activities on Earth includes the disruptions to communications and energy infrastructures, increased radiation to objects in near earth space, and random incursions within circuits and logic processors, which would produces undesired logic outputs and readings due to single event offsets [2]. And given the distance between the Earth and the Sun, efforts has been made to use machine learning to analyse the images of solar atmosphere, from which solar flares can be predicted [3].

This data analysis project will investigate the role of plasma and gyrosynchrotron emissions in the context of microwave bands. The project will begin with theoretical backgrounds on plasma emissions and gyrosynchrotron emissions, in terms of their mathematical models and how the emissions are considered within different frequency ranges. The project will then move on to give detailed rundown on the computational packages used to perform the data analysis, with generated results and plots explained. Afterwards, the project investigates the importance of plasma emission modeling in gyrosynchrotron emission curve fitting.

## II. Theoretical background

This project investigates the radio signal data sourced from NoRP and RSTN sites, and a total of three data sequences are considered. The two sequences from RSTN is obtained from Learmonth site (apl) in Australia and Palehua site (phf) in Hawaii. The general frequency of interest lies between 0.245 GHz to 80 GHz. The three radio telescope sites are selected in a way to provide a wide coverage. The three sites effectively forms a equalateral triangle, with the peak flare emission event time corresponding to the local time at each sites at around noon. Therefore, the inclusion of other sites, where the sun is not in line of sight with the telescopes would not be productive.

## A. Gyrosynchrotron emissions

The primary emission is modeled as Gyrosynchrotron (gyro) emissions. It is shown that the gyro emissions can be considered as the dominating means of emissions for solar flares at relatively higher frequencies [4]. That is, when examining the emission profile of the energy released during the said flaring events, a peak emission frequency can usually be located at around 10 GHz and the emissions gradually decays onto higher frequencies [5]. For the purpose of this data analysis project, we define a low frequency threshold at 2 GHz.

A non-linear relation between the flux  $\Phi$  and frequency  $v$  can be modeled with some fit parameter  $A, B, a, b$  as follows:

$$\Phi = A \cdot v^a \cdot [1 - \exp(-B \cdot v^{-b})] \quad (1)$$

In this simplified model, we have low frequency slope  $a$  and high frequency slope  $a - b$ , indicating the behaviour of gyro emissions as seen in lower and higher frequency ranges respectively. It is noted that the simplified model shown in the above equation is relatively immune to data noise [4].

## B. Plasma emissions

In lower frequencies, the received radio data can be interpreted as a power law, as seen in the equation below:

$$\Phi = c \cdot v^k \quad (2)$$

Consider the general shape of a power law curve with negative  $k$ , the plasma emission profiles would dominate the lower frequencies while being relatively trivial at higher frequency ranges, where the gyro emission starts dominating. As a result, when slicing the dataset for curve fitting, we only fit the plasma model for lower frequencies, rather than using the entire data array to construct a plasma emission profile.

In theory, a partial data fit with only higher frequency dataset used for gyro emissions fitting would produce a similar result as the dataset with plasma emission model corrected. However, it is worth noticing that a complete plasma emission model along all frequency entries may provide a better adjustment to the original dataset, and consequently producing better gyro emission fits.

### III. Computational methods

This project is programmed under python environment and powered by Anaconda. We use a working folder to organize the data repository, the media output, and the working python files within VSCode. In this section, we introduce the workflow and the functions within the modules. Then, we provide the overall logic of the master file, from which the generated results will be discussed in Sec. IV. The full project repository can be found on the relevant [GitHub page](#).

#### A. Module, structure, and workflow

The primary library used in this project are `numpy`, `scipy`, and `matplotlib`. Separate libraries such as `scienceplots` and `datetime` are also used to provide plotting and data conversion support.

The radio data files are stored under `./data/` in the local directory, with media output stored under `./media/`. A master script is used to get the methods constructed within other modules files. The module files are prefixed with `data_` and the master script is named `main.py`. Additionally, a separate module file called `data_legacy.py` is used to deposit legacy plotter functions.

This project employs the use of VSCode for text editing, with the modifications pushed to remote git repository using integrated bash support. Within each module file, the code blocks are constraint within separate functions, such that the master script file is able to read, assign, and pass on the variables with function calls only.

#### B. Data reader module

We construct a customized module, aliased `data_reader.py`, for handling initial `.csv` file content reading, formatting, and writing. During the process, specific attention is needed for the `dtype` of the given data.

For the given NoRP and RSTN dataset, the files containing time data need to be processed into `numpy.array()`. That is, by considering the given data structure, with the first column containing the number of milliseconds from the start of a day and the second column being the number of days since January 01, 1979, we employ `datetime` module to calculate and convert the two columns into a single entry along an array, where the output become a time string formatted as “YYYY-MM-DD hh:mm:ss”. In effect, we loaded the time arrays as unassigned 64-bit integers (`numpy.uint64`) to avoid overflow.

#### C. Data handler module

##### 1. Function: `loader()`

We start by importing the `data_reader` module into the current module. Consequently, a new function could be constructed to take in the data path of the data files and generate the formatted `numpy.array()` tuples in the return.

One thing to pay attention is that the shape of the data array must be considered to make sure that the

dimensionality of the data is consistent with the other data files.

##### 2. Function: `validator()`

Within the original dataset, a validity checker file is included to act as a filter for dealing with invalid data recordings received on the sensors. Thus, a validator function was constructed to first initiate a data mask from the validity data.

The mask is chosen such that all element on the given row must all be true when logically summed up. Afterwards, the mask is passed onto the flux and time data arrays to filter out invalid flux arrays and its corresponding time array.

##### 3. Function: `quiet_sun()`

Another function within the `data_handler` module performs the quiet sun flux calculation. This function takes in a tuple of data arrays that requires the calculation, and generates the corresponding quiet sun flux subtracted results tuple. The operation involves in taking the mean value along a column and have all elements along the said column subtract the calculated mean value. The operation is then looped over all columns within a given `numpy.array()` and then over all arrays within the tuple.

Now, this calculation method is based on the overall readings of the flux emission over a selected period of time, that is, a longer time sequence of flux readings could provide a more accurate average flux emission value array on each frequencies, compared to an average flux value array calculated from a limited time sequence. This is due to the drastic increase in flux intensity at flare event, where the recorded flux values would be of several orders of magnitudes higher than the baseline value.

Therefore, an alternative of calculating the quiet sun flux value would be to either: 1) Obtain the annual flux readings and perform the same operations as seen above on the annual flux data; 2) To apply the negative value of the fitted gyro emission profile to the recorded flux array first, before performing the quiet sun calculations; 3) By slicing the original data array to eliminate the time sequence around the peak emission time, such that only the baseline readings are considered for quiet sun calculations.

##### 4. Function: `collector()`

At the end, a more complex function is created to extract out, combine, and generate the combined flux and frequency array for all three dataset at the solar flare event peak time. This is done by first construct a sub-function called `peak_time()` that identifies the array row index of the peak time, from which the indexes are passed on to two other sub-functions, `peak_freq()` and `peak_flux()`, which would index out, and concatenate the arrays together into a single output.

The master function then calls to use these aforementioned sub-functions to get the combined flux and frequency array data. Notice that the frequency data array

is then made unique, with non-one-to-one mapped flux data averaged. The results are then returned as a tuple.

#### D. Data fitter module

The `data_fitter` module utilizes `numpy`, `curve_fit` from `scipy.optimize`, and `chi2` from `scipy.stats`. A total of six functions are defined, two for fit model definition, one for getting the fitted equations' expressions for plot labeling, two more for the curve fits themselves, and one more for denoising the flux array data.

1. *Function: `gyro_model()`, `plas_model()`, and `fit_label()`*

The two model functions takes in the parameter arguments and return the fit functions as a direct callable. The labeling generator function unpacks the fitted results and return a **f-string** formatted entry tuple to be passed onto future plotters.

2. *Function: `gyro_fitter()` and `plas_fitter()`*

The two fitter functions shares similar logic in taking the flux and frequency `numpy.array()` as data. The functions run the data through the `curve_fit()` method to obtain the fit parameter tuple and its covariance matrix.

The residual term is then calculated for chi-squared statistics, where the chi-squared value is obtained with `chi2.cdf()`. Additionally, a p-value is also calculated with the degree of freedom selected from the number of data point entries and the number of fit parameters.

The two fit functions differs from results printout sections and their initial fit parameter guesses. As the power law fit function contains fewer parameters than the gyro model.

3. *Function: `gyro_pass()`*

The denoiser function serves to pass the combined, made unique flux and frequency data coordinates through the influence of a fitted plasma emission model in lower frequencies. That is, the fitted plasma model generates a flux data array, which would be used to adjust the original combined flux array, such that the plasma emission is effectively denoised from the original data.

The function slice the array by a cut-off threshold defined by the user at 2 GHz, pass the plasma model data adjustment, and then recombined the data arrays together to form the function return.

#### E. Data plotter module

A total of six figures are generated by the plotter module, with one having only valid NoRP data around the peak time, the other containing only RSTN data around the peak time. Additionally, a combined plot with NoRP and RSTN data is generated to show the flux time evolution around peak time. Then, plots containing only the peak time flux array of NoRP and RSTN is generated, along with the combined, made unique flux against frequency plot that also ships with initial `curve_fit()` results. The final plot shows a corrected flux-frequency plot with fitted curves.

By invoking the use of `scienceplots` package, we are able to globally define the plot style for `matplotlib.pyplot` such that then generated plot follows the use of L<sup>A</sup>T<sub>E</sub>X in a Jupyter notebook style output. For each plotter function, a log-log scale is used for the axis scale options.

1. *Function: `plot_generator()`*

The plot generator function takes in the tuple of all the plotting argument tuples and call their respective plotter functions. This is to reduce the module import in the master file.

### IV. Analysis

In this section, we will start by providing the master python file execution logic. Then, the curve fit results are shown with generated plots.

#### A. Master file logic

The master script utilize the module files and import them into the script first. Before calling upon any functions, the master file defines the key variables such as local data file paths and the peak time formatted string. This step can be further optimized by passing in arguments that uses user inputs to determine the local file paths. Then, the master file calls upon the `loader()` function to deposit all the working variables into the local repository.

The filtered data is then fetched by calling upon the `validator()` function, with the quiet sun flux array calculated results deposited by invoking the `quiet_sun()` function. That is, at this stage, the master file has completed the preliminary data preparations for NoRP event analysis.

Now, the argument tuples for sub functions are created and packed into a master argument tuple, which would be passed on to the `collector()` function to generate the combined frequency and flux array at the peak radio emission time by unpacking the argument in the function call.

Effectively, the master file has now completed preparing the data arrays to be used for curve fitting. By defining a low frequency threshold, the curve fit results would be assigned to local variables by making calls on functions of `gyro_fitter()` and `plas_fitter()`. At this stage, the combined flux data array is adjusted to the fitted plasma emission model with `gyro_pass()` function call.

And finally, a tuple of plotter resources are built by rounding up the local variables. The argument tuple is then passed into `plot_generator()` to get the plots saved under local media directory.

#### B. Flux against frequency plots

The individual time evolution of flux data at around the purposed peak time for NoRP and RSTN sites are listed. We observe in Fig. 1 and Fig. 2 that the flux recordings around the peak time shows a consistent peak flux value at around 10 GHz, with lower frequency emissions being relatively negligible on the

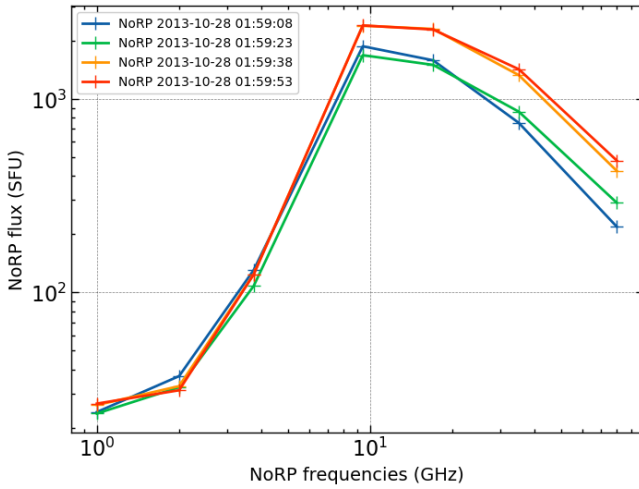


FIG. 1. Averaged flux against frequency plot for NoRP site. Flux calculated at  $\pm 30$  s around labeled time sequence. Flux sequences plotted around peak time on 2013-10-28 at 01:59:38 UTC.

NoRP dataset. However, it is discovered that for RSTN datasets, at lower emission frequencies, a significant amount of plasma emissions were recorded.

Additionally, plots are combined for NoRP and RSTN sites results. The combined plot as seen in Fig. 3 follows the observations from previous plots, that the data recorded from these three sites follows along with each other and forms a mutual agreement on their trends. Furthermore, the three sets of data plotted at the peak emission time suggested by NoRP data, as seen in Fig. 4 give an insight that the data set could be combined together to extend their individual frequency range.

At the end are the combined plots for NoRP and RSTN sites, the frequency and flux array were combined into a singular array respectively. The fitted curves are addressed and labeled on the plots. we now observe a clear association in the plasma emission curve fit at lower frequencies. However, the two gyro fits, both before and after the plasma emission model were considered, displays little difference in the general geometry of the fitted curve. It is worth noticing that the  $\chi^2$  test shows that, with the flux array getting adjusted with the plasma emission model, the  $\chi^2$  value decreased drastically, indicating a much better fit of the gyro model to the given combined dataset.

### C. Curve fit results

The curve fitting results are printed in the terminal as listed below, the initial gyro fit with the plasma emission profile included in the data shows a low frequency slope at  $a = 2.570$  and a high frequency slope at  $a - b = -1.140$ :

```
Gyro fitter results Initial
=====
Gyro fitter fitted parameters
A:                9.5675
B:                7189.8
```

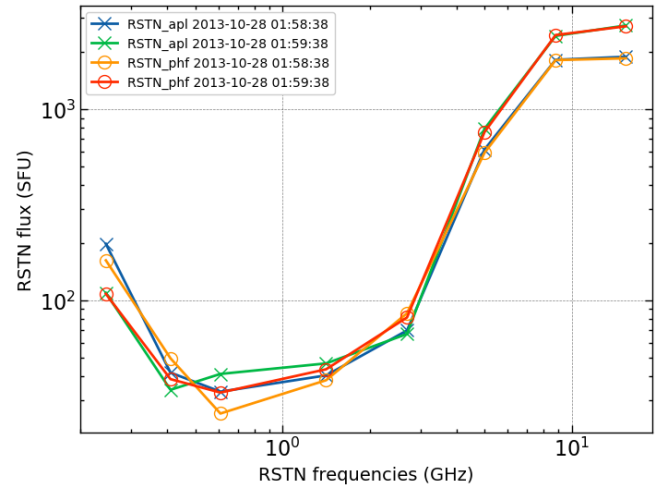


FIG. 2. Flux against frequency plot for RSTN sites. Flux calculated at  $\pm 30$  s around labeled time sequence. Flux sequences plotted around peak time on 2013-10-28 at 01:59:38 UTC.

```
a:                2.5703
b:                3.71
Low freq slope:   2.5703
High freq slope:  -1.1397
```

#### Chi-square test result

```
Chi-square:       6.0352e+05
p-value:          0
=====
```

The initial fit generates a  $\chi^2$  value, which shows a bloated number at  $\chi^2 = 6.03^5$ . This provide a baseline for future fits, as higher  $\chi^2$  value, when calculated from the following method, indicates a non-optimized fit between the model and the dataset. A p-value could also be calculated. However, given the bloated  $\chi^2$  results, the associated p-value would not be of help in determining the goodness of fit in the specific case.

$$\chi^2 = \sum \left( \frac{(y_{\text{data}} - y_{\text{model}})^2}{y_{\text{model}}} \right) \quad (3)$$

The plasma emission model as fitted for lower frequencies gives the correction model for flux data entries under 2 GHz.

#### Plas fitter results

```
=====
Plas fitter fitted parameters
c:                2.8574
k:                -3.4994
```

#### Chi-square test result

```
Chi-square:       2319.9
p-value:          0
=====
```

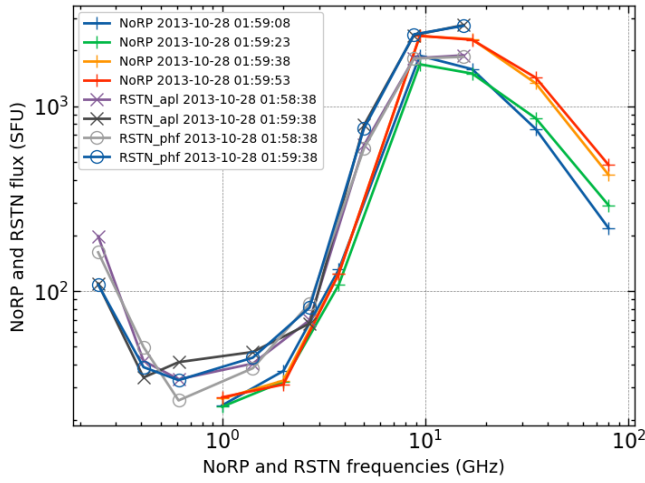


FIG. 3. Averaged flux against frequency plot for NoRP and RSTN sites. Averaged flux calculated at  $\pm 30$  s around labeled time sequence. Flux sequences plotted around peak time at 2013-10-28 at 01:59:38 UTC.

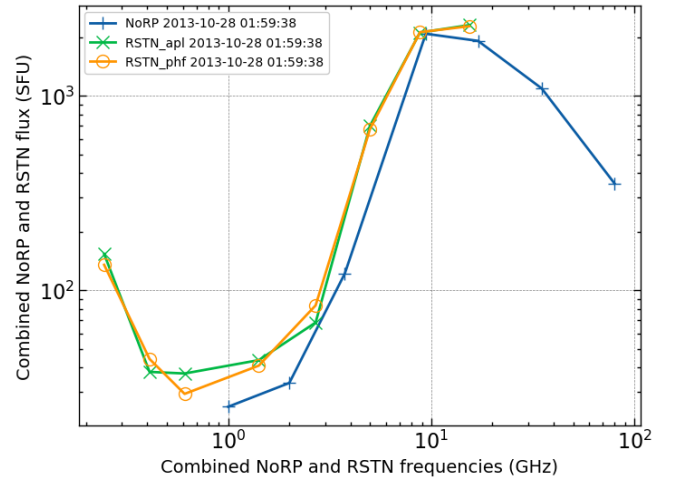


FIG. 4. Flux against frequency plot for RSTN sites. Averaged flux calculated at  $\pm 30$  s around the peak time sequence. Flux sequences plotted following the peak time at 2013-10-28 at 01:59:38 UTC.

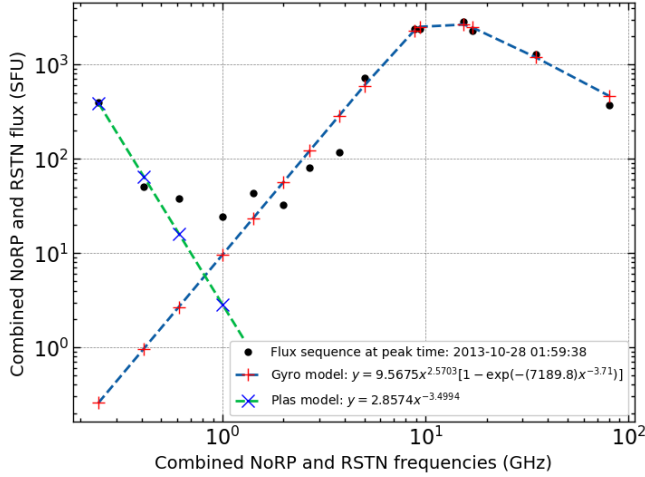


FIG. 5. Flux against frequency plot for NoRP and RSTN sites. Flux calculated at  $\pm 30$  s around labeled time sequence and combined into singular sequence. The frequencies on the frequency array are unique and maps to the flux array value one-on-one. Combined and averaged flux sequence plotted around peak time on 2013-10-28 at 01:59:38 UTC.

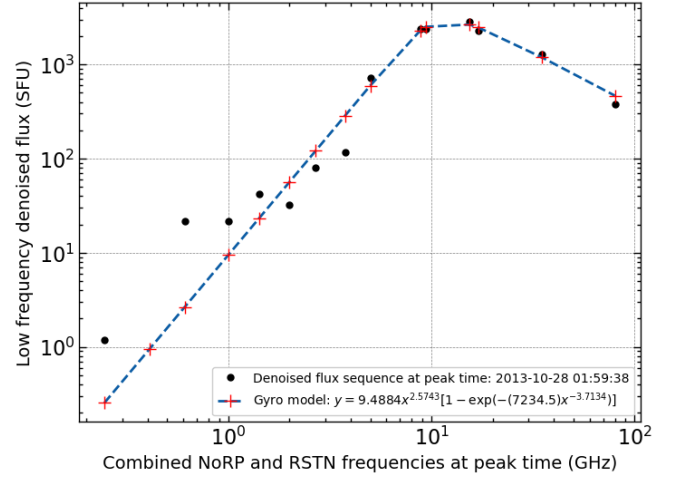


FIG. 6. Flux against frequency plot for NoRP and RSTN sites. Flux calculated at  $\pm 30$  s around labeled time sequence and combined into singular sequence. The frequencies on the frequency array are unique and maps to the flux array value one-on-one. The combined flux array is now denoised by adding the negative value of the fitted plasma emission results. Combined and averaged flux sequence plotted around peak time on 2013-10-28 at 01:59:38 UTC.

In Fig. 5, consider a plasma emission model for all frequency entries, we should expect a higher coefficient of  $c$ . By eliminating the plasma emission profile from the original combined flux curve, we refit the data and get:

```
Gyro fitter results Denoised
=====
Gyro fitter fitted parameters
A:          9.4884
B:          7234.5
a:          2.5743
b:          3.7134
```

```
Low freq slope:      2.5743
High freq slope:    -1.1391
```

```
Chi-square test result
Chi-square:         622.28
p-value:           0
=====
```

The new fit illustrates a slightly different low and high frequency slope combination, with low frequency slope  $a = 2.574$  and high frequency slope  $a - b = -1.1391$ . That is, the general trend of the fit is not affected by the



inclusion of the plasma emissions model. This could be attributed to the fact that the higher flux entries are concentrated at higher frequencies, with a theorized peak at 10 GHz, effectively rendering the low frequency, plasma emission entries, as the outlier of the dataset.

However, it is worth noticing that the  $\chi^2$  value of the new fit has been reduced by a few orders of magnitude, at  $\chi^2 = 6.62^2$ . This indicates that the plasma emission model is still significant in providing “noise ” to the gyro model fit, and denoising the data with the plasma emission could help to clean up the data to provide a better gyro fit. This is not to say that the plasma emission is generically significant to the dataset itself. However, we do recognize that the adjustment is necessary in producing better model fits. With the final fit equation at:

$$\begin{aligned} \Phi &= A \cdot v^a \cdot [1 - \exp(-B \cdot v^{-b})] \\ &= 9.4884 \cdot v^{2.5743} \cdot [1 - \exp(-7234.5 \cdot v^{-3.7134})] \end{aligned} \quad (4)$$

Consequently, by taking the first order derivative of the expression, we are able to discover that a peak emission frequency of the fitted gyro emission model resides

around  $v \simeq 12.084$  GHz. This peak emission frequency is consistent with the theory prediction at the magnitude of  $10^1$  GHz.

## V. Conclusion

This radio data analysis projects have processed the radio data obtained from NoRP and RSTN sites regarding a peak solar flare emission event. With the generated flux vs. frequency plots and their associated curve fitting results, this project recognizes the importance of plasma emission model in its influence to gyro emissions across all frequencies of the microwave band entries. It is understood that the fitting of plasma emission model could help with denoising the received flux data array, and consequently generate better curve fitting results for gyro emissions. The computational methodology is listed and explained to provide clear guidance for repeating the data analysis steps. At the end, a non-linear least square fit of gyro emission was performed on the processed dataset, where the plasma emission profile is removed from the original dataset. The final curve fit displayed a better alignment to the dataset as seen from the  $\chi^2$  value calculations.

- 
- [1] R. Schwenn, *Living Reviews in Solar Physics* **3**, 2 (2006).
  - [2] D. N. Baker, E. Daly, I. Daglis, J. G. Kappenman, and M. Panasyuk, *Space Weather* **2** (2004).
  - [3] T. Colak and R. Qahwaji, *Space Weather* **7** (2009).
  - [4] M. Stahli, D. E. Gary, and G. J. Hurford, *Solar Physics* **120**, 351 (1989).
  - [5] K. L. Klein, *Astronomy and Astrophysics* **183**, 341 (1987).

## A. Code file: README.md

```
1 ## Radio data analysis
2 This is a course project for *ASTRO-5010* at *University of Glasgow*.
3 - Drafted: Mar 15, 2023
4 - Edited: Mar 31, 2023
5
6 ## Data set
7 We employ the data obtained from:
8 1. NoRP: Nobeyama Radio Polarimeters
9 2. RSTN: Radio Solar Telescope Network
10
11 ## Purpose
12 We conduct basic level data analysis and plotting to understand:
13 1. Gyrosynchrotron emissions
14 2. Plasma emissions
15
16 ## Steps
17 We will perform the tasks with the following steps:
18 1. Load NoRP data set
19 2. Convert time array data to standard format
20 3. Filter out valid data readouts
21 4. Filter out quiet sun noise at assigned freq
22 5. Plot freq based flux data against time
23 6. Load RSTN data
24 7. Fit gyrosynchrotron emission
25 8. Fit plasma emission
26
27 ## Working file structure
28 Main file:
29 - main.py
30
31 Data reader modules:
32 - data_reader.py
33
34 Data processing modules:
35 - data_handler.py
36
37 Data fitter modules:
38 - data_fitter.py
39
40 Plotter modules:
41 - data_plotter.py
42
43 ## Supplementary files
44 Readme:
45 - Readme.md
46
47 Legacy plotter repo:
48 - data_legacy.py
49
50 ## Folder structure
51 Data folder:
52 - data
53
54 Plot folder:
55 - media
```

## B. Code file: main.py

```

1  """
2  This is the master script of the radio data analysis project.
3
4  Created on Wed Mar 15 2023
5
6  @author: Yang-Taotao
7  """
8  # %% Library import
9  # Custom module import
10 # Data handler import
11 from data_handler import loader, validator, quiet_sun, collector
12
13 # Data plotter import
14 from data_plotter import plot_generator, log_avg_plotter
15
16 # Data fitter import
17 from data_fitter import gyro_fitter, plas_fitter, gyro_pass
18
19 # %% Data path and peak time assignment
20 # Assign norp, apl, and phf file path
21 data_path = ("./data/norp/", "./data/apl/", "./data/phf/")
22 # Assign the peakttime of flux recording
23 norp_peak_time = "2013-10-28 01:59:38"
24
25 # %% Load csv data into data repo
26 # Deposit data arrays
27 data_repo = loader(data_path)
28 # Assign loaded data to variables
29 (
30     norp_fi,
31     norp_freq,
32     norp_mvd,
33     norp_tim,
34     apl_fi,
35     apl_freq,
36     apl_tim,
37     phf_fi,
38     phf_freq,
39     phf_tim,
40 ) = [data_repo[i] for i in range(len(data_repo))]
41
42 # %% NoRP validity filter result deposit
43 norp_tim_valid, norp_fi_valid = validator(norp_mvd, norp_tim, norp_fi)
44
45 # %% NoRP quiet sun result deposit
46 # Generate data array tuple
47 quiet_sun_data = (norp_fi_valid, apl_fi, phf_fi)
48 # Deposit quiet sun results
49 norp_fi_peak, apl_fi_peak, phf_fi_peak = quiet_sun(quiet_sun_data)
50
51 # %% Peak time array argument assignment
52 # Time, Freq, Flux sub function arguemnt repo
53 arg_time, arg_freq, arg_flux = (
54     # arg_time
55     (
56         norp_tim_valid,

```



```

57     apl_tim,
58     phf_tim,
59     norp_peak_time,
60 ),
61 # arg_freq
62 (
63     norp_freq,
64     apl_freq,
65     phf_freq,
66 ),
67 # arg_flux
68 (
69     norp_fi_peak,
70     apl_fi_peak,
71     phf_fi_peak,
72 ),
73 )
74 # Peak time argument tally
75 peak_arg = (arg_time, arg_freq, arg_flux)
76
77 # %% Get peak time combined flux and time array
78 peak_time_freq, peak_time_flux = collector(*peak_arg)
79
80 # %% Curve fitter key value assignment - Define cut-off freq
81 freq_cut = 2
82
83 # %% Generate combined fit results
84 # Get fitted results
85 results_gyro, results_plas = (
86     gyro_fitter(peak_time_freq, peak_time_flux, "Initial"),
87     plas_fitter(peak_time_freq, peak_time_flux, freq_cut),
88 )
89 # Assign fit parameters
90 gyro_param, plas_param = results_gyro[0], results_plas[0]
91
92 # %% Flux array denoise at low freq
93 gyro_freq_denoise, gyro_flux_denoise = (
94     peak_time_freq,
95     gyro_pass(peak_time_freq, peak_time_flux, freq_cut, plas_param),
96 )
97
98 # %% Refit with denoised data
99 results_denoise = gyro_fitter(gyro_freq_denoise, gyro_flux_denoise, "Denoised")
100 denoise_param = results_denoise[0]
101
102 # %% Plotter argument assignment
103 # Plot arguments assignment
104 plt_arg1, plt_arg2, plt_arg3, plt_arg4, plt_arg5 = (
105     # NoRP plotter arguments
106     (norp_tim_valid, norp_fi_peak, norp_freq, norp_peak_time),
107     # RSTN plotter arguments
108     (
109         apl_tim,
110         phf_tim,
111         apl_fi_peak,
112         phf_fi_peak,
113         apl_freq,
114         phf_freq,

```

```

115     norp_peak_time ,
116 ),
117 # Combined plotter arguments
118 (
119     norp_tim_valid ,
120     apl_tim ,
121     phf_tim ,
122     norp_fi_peak ,
123     apl_fi_peak ,
124     phf_fi_peak ,
125     norp_freq ,
126     apl_freq ,
127     phf_freq ,
128     norp_peak_time ,
129 ),
130 # Peak time combined plotter arguments
131 (
132     peak_time_freq ,
133     peak_time_flux ,
134     norp_peak_time ,
135     gyro_param ,
136     plas_param ,
137     freq_cut ,
138 ),
139 # Denoised plotter arguments
140 (
141     gyro_freq_denoise ,
142     gyro_flux_denoise ,
143     norp_peak_time ,
144     denoise_param ,
145     plas_param ,
146 ),
147 )
148
149 # %% Optional - Averaged flux array parser
150 norp_peak_avg, apl_peak_avg, phf_peak_avg = log_avg_plotter(plt_arg3)
151
152 # %% Plot generator argument assignment
153 plt_arg = (plt_arg1, plt_arg2, plt_arg3, plt_arg4, plt_arg5)
154
155 # %% Plot generation
156 plot_generator(plt_arg)

```

## C. Code file: data\_reader.py

```

1  """
2  This is the data loader script of the radio data analysis project.
3
4  Created on Wed Mar 15 2023
5
6  @author: Yang-Taotao
7  """
8  # %% Library import
9  # Library import
10 import datetime as dt
11 import numpy as np
12
13 # %% Data parser
14 # CSV data parser
15 def csv_loader(file_path, dtype=float):
16     """
17     Parameters
18     -----
19     file_path : string
20         Path to data file folder.
21     dtype : dtype, optional
22         The dtype of assigned file. The default is float.
23
24     Returns
25     -----
26     data : array
27         The data readout array.
28
29     """
30     # Initial CSV data load with specified dtype
31     data = np.loadtxt(file_path, delimiter=",", dtype=dtype)
32
33     # Time data loader and convertor
34     if dtype == np.uint64:
35         # Define start point of datetime at 1979-01-01 as day01
36         time_origin = dt.datetime(1979, 1, 1) - dt.timedelta(days=1)
37
38         # Construct time function for calculating time result
39         time_update = [
40             time_origin + dt.timedelta(milliseconds=ms + days * 86400000)
41             for ms, days in data # col0, col1: ms, days
42         ]
43
44         # Combined datetime data
45         data = np.array(
46             [time.strftime("%Y-%m-%d %H:%M:%S") for time in time_update]
47         )
48
49         # Return converted time data array
50         return data
51
52     # Data loader for all other dtype
53     return data

```

## D. Code file: data\_handler.py

```

1  """
2  This is the data handler script of the radio data analysis project.
3
4  Created on Wed Mar 15 2023
5
6  @author: Yang-Taotao
7  """
8  # %% Library import
9  # Library import
10 import numpy as np
11 from data_reader import csv_loader
12
13 # %% Data loader
14 # Combined data loader
15 def loader(data_path):
16     """
17     Parameters
18     -----
19     data_path : tuple
20         Tuple of data folder path.
21
22     Returns
23     -----
24     result : tuple
25         Tuple of loaded data arrays.
26     """
27     # Local path variable repo
28     flux, freq, mvd, tim = ("flux.csv", "freq.csv", "mvd.csv", "tim.csv")
29
30     # Pack data into tuple
31     result = (
32         csv_loader(data_path[0] + flux),
33         csv_loader(data_path[0] + freq),
34         csv_loader(data_path[0] + mvd, dtype=int),
35         csv_loader(data_path[0] + tim, dtype=np.uint64),
36         csv_loader(data_path[1] + flux).transpose(),
37         csv_loader(data_path[1] + freq),
38         csv_loader(data_path[1] + tim, dtype=np.uint64),
39         csv_loader(data_path[2] + flux).transpose(),
40         csv_loader(data_path[2] + freq),
41         csv_loader(data_path[2] + tim, dtype=np.uint64),
42     )
43
44     # Return the assignment
45     return result
46
47
48 # %% Data validator
49 # NORP data filter based on mvd file
50 def validator(data_norp_mvd, data_norp_tim, data_norp_fi):
51     """
52     Parameters
53     -----
54     data_norp_mvd : array
55         Validity array of NORP data.
56     data_norp_tim : array

```

```

57     Time array of NORP data.
58     data_norp_fi : array
59     Flux array of NORP data.
60
61     Returns
62     -----
63     data_norp_tim_valid : array
64         Valid time array of NORP data.
65     data_norp_fi_valid : array
66         Valid flux array of NORP data.
67     """
68     # Generate valid data mask based on boolean readout over single rows
69     data_norp_mask = np.all(data_norp_mvd.astype(bool), axis=1)
70
71     # Filter the time and flux data through mask
72     data_norp_tim_valid, data_norp_fi_valid = (
73         data_norp_tim[data_norp_mask],
74         data_norp_fi[data_norp_mask],
75     )
76
77     # Return filtered result
78     return (data_norp_tim_valid, data_norp_fi_valid)
79
80
81 # %% Quiet sun flux calculator
82 # Quiet sun calculator
83 def quiet_sun(data_array_tuple):
84     """
85     Parameters
86     -----
87     data_array_tuple : tuple
88         Flux array tuple.
89
90     Returns
91     -----
92     data_array_repo : tuple
93         Filtered quiet sun array data tuple.
94     """
95     # Loop through the arrays to generate quiet sun flux array tuple
96     data_array_repo = tuple(
97         array - np.mean(array, axis=0) for array in data_array_tuple
98     )
99
100     # Return quiet sun flux array tuple
101     return data_array_repo
102
103
104 # %% Peak time array collector
105 # Peak time array collector
106 def collector(arg_time, arg_freq, arg_flux):
107     """
108     Parameters
109     -----
110     arg_time : tuple
111         Tuple of time and peak time data arrays.
112     arg_freq : tuple
113         Tuple of freq data arrays.
114     arg_flux : tuple

```

```

115     Tuple of flux data arrays.
116
117 Returns
118 -----
119 data_fi_combined : array
120     Peak time flux array combined.
121 data_freq_combined : array
122     Peak time freq array combined.
123 """
124 # Peak time index identifier and freq combiner
125 def peak_time(arg_time):
126     """
127     Parameters
128     -----
129     arg : tuple
130         Tuple of time data arrays with peak time.
131
132     Returns
133     -----
134     idx_norp : integer
135         Index of NoRP peak time.
136     idx_apl : integer
137         Index of apl peak time.
138     idx_phf : integer
139         Index of phf peak time.
140     data_freq_peak_time_combined : array
141         Combined freq array.
142     """
143     # Local variable repo
144     (
145         data_norp_tim_valid,
146         data_apl_tim,
147         data_phf_tim,
148         data_norp_peak_time,
149     ) = [arg_time[i] for i in range(len(arg_time))]
150
151     # Index locator
152     idx_norp, idx_apl, idx_phf = (
153         np.where(data_norp_tim_valid == data_norp_peak_time)[0][0],
154         np.where(data_apl_tim == data_norp_peak_time)[0][0],
155         np.where(data_phf_tim == data_norp_peak_time)[0][0],
156     )
157
158     # Return index repo
159     return (idx_norp, idx_apl, idx_phf)
160
161 # Peak time freq collector
162 def peak_freq(arg_freq):
163     """
164     Parameters
165     -----
166     arg_freq : tuple
167         Tuple of freq data arrays.
168
169     Returns
170     -----
171     data_freq_combined : array
172         Combined freq array.

```



```

173     """
174     # Local variable repo
175     data_norp_freq, data_apl_freq, data_phf_freq = [
176         arg_freq[i] for i in range(len(arg_freq))
177     ]
178
179     # Concatenate freq array
180     data_freq = tuple(
181         [
182             data_norp_freq,
183             data_apl_freq,
184             data_phf_freq,
185         ]
186     )
187     data_freq_combined = np.concatenate(data_freq)
188
189     # Return combined freq array
190     return data_freq_combined
191
192 # Peak time flux collector
193 def peak_flux(arg_flux):
194     """
195     Parameters
196     -----
197     arg_flux : tuple
198         Tuple of flux data arrays.
199
200     Returns
201     -----
202     data_flux_combined : array
203         Combined flux array.
204
205     """
206     # Local variable repo
207     data_norp_flux_peak, data_apl_flux_peak, data_phf_flux_peak = [
208         arg_flux[i] for i in range(len(arg_flux))
209     ]
210
211     # Import peak identifier result
212     idx_norp, idx_apl, idx_phf = peak_time(arg_time)
213
214     # Pick peak time flux array out
215     data_norp_flux_peak, data_apl_flux_peak, data_phf_flux_peak = (
216         data_norp_flux_peak[idx_norp],
217         data_apl_flux_peak[idx_apl],
218         data_phf_flux_peak[idx_phf],
219     )
220
221     # Concatenate freq array
222     data_flux = tuple(
223         [
224             data_norp_flux_peak,
225             data_apl_flux_peak,
226             data_phf_flux_peak,
227         ]
228     )
229     data_flux_combined = np.concatenate(data_flux)
230

```

```
231     # Return combined freq array
232     return data_flux_combined
233
234 # Peak time freq, flux array generator
235 data_freq_combined, data_flux_combined = peak_freq(arg_freq), peak_flux(
236     arg_flux
237 )
238
239 # Array sorter
240 # Get numpy index sort array
241 idx_sort = np.argsort(data_freq_combined)
242 # Sort array with index
243 data_freq_sorted, data_flux_sorted = (
244     data_freq_combined[idx_sort],
245     data_flux_combined[idx_sort],
246 )
247
248 # Array unique value filter
249 # Get the unique freq values array
250 data_freq_final = np.unique(data_freq_sorted)
251
252 # Calculate the mean y values for each unique x value
253 data_flux_final = np.array(
254     [
255         np.mean(data_flux_sorted[data_freq_sorted == xval])
256         for xval in data_freq_final
257     ]
258 )
259
260 # Return combined peak time flux array
261 return (data_freq_final, data_flux_final)
```

## E. Code file: data\_fitter.py

```

1  """
2  This is the data fitter script of the radio data analysis project.
3
4  Created on Wed Mar 15 2023
5
6  @author: Yang-Taotao
7  """
8  # %% Library import
9  # Library import
10 import numpy as np
11 from scipy.optimize import curve_fit
12 from scipy.stats import chi2
13
14 # %% Fit model definition
15 # Gyro model definition
16 def gyro_model(x_val, param_a_cap, param_b_cap, param_a, param_b):
17     """
18     Parameters
19     -----
20     x_val : array
21         Gyro model x value array.
22     param_a_cap : float
23         Fit param A.
24     param_b_cap : float
25         Fit param B.
26     param_a : float
27         Fit param a.
28     param_b : float
29         Fit param b.
30
31     Returns
32     -----
33     array
34         Gyro model.
35     """
36     # Return gyro model
37     return (
38         param_a_cap
39         * (x_val**param_a)
40         * (1 - np.exp(-param_b_cap * (x_val ** (-param_b))))
41     )
42
43 # Plas model definition
44 def plas_model(x_val, param_c, param_k):
45     """
46     Parameters
47     -----
48     x_val : array
49         Plasma model x value array.
50     param_c : float
51         Fit param c.
52     param_k : float
53         Fit param k.
54
55     Returns
56

```

```

57     -----
58     array
59         Plasma model.
60
61     """
62     # Return plas model
63     return param_c * (x_val**param_k)
64
65
66 # %% Fitted function result label generator
67 def fit_label(gyro_param, plas_param):
68     """
69     Parameters
70     -----
71     gyro_param : tuple
72         Gyro model fit param tuple.
73     plas_param : tuple
74         Plas model fit param tuple.
75
76     Returns
77     -----
78     label_gyro : string
79         Fitted gyro model expression.
80     label_plas : string
81         Fitted plas model expression.
82     """
83     # Cache into singular tuple
84     fit_param = np.concatenate([gyro_param, plas_param]).ravel()
85
86     # Local fit variable unpack
87     fit_a_cap, fit_b_cap, fit_a, fit_b, fit_c, fit_k = [
88         fit_param[i] for i in range(len(fit_param))
89     ]
90
91     # Label generator
92     label_gyro, label_plas = (
93         # Gyro model
94         rf"Gyro model: $y={fit_a_cap:.5g}x^{{{fit_a:.5g}}}"
95         rf"[1-\exp({{-(fit_b_cap:.5g)}x^{{{fit_b:.5g}}}})]$",
96         # Plas model
97         rf"Plas model: $y={fit_c:.5g}x^{{{fit_k:.5g}}}$",
98     )
99
100     # Result return
101     return (label_gyro, label_plas)
102
103
104 # %% Gyro fitter
105 # Gyro fitter function
106 def gyro_fitter(data_freq, data_flux, title):
107     """
108     Parameters
109     -----
110     data_freq : array
111         Combined freq data array.
112     data_flux : array
113         Combined flux data array.
114     title : string

```

```

115         Additional title for plot customization.
116
117 Returns
118 -----
119 params : array
120         Fit parameters array.
121 cov : float
122         Covariance matrix of the fit.
123 chi_sqr :float
124         Chi-squared value of the fit.
125 chi_p_val : float
126         The p-value from the chi-squared test.
127 """
128 # Generate filtered data
129 data_x, data_y = (
130     data_freq,
131     data_flux,
132 )
133
134 # Initial parameter guess
135 param_guess = [1, 1, 1, 1] # param_A, param_B, param_a, param_b
136
137 # Curve fit results
138 params, cov = curve_fit(gyro_model, data_x, data_y, p0=param_guess)
139
140 # Residuals generator
141 # Get fitted model
142 fit_model = gyro_model(data_x, *params)
143 # Residual generator
144 fit_resid = data_y - fit_model
145
146 # Chi2 Tester
147 # Chi2 calculation and dof generation
148 chi_sqr, chi_dof = (
149     np.sum(fit_resid**2 / fit_model),
150     len(data_x) - len(params),
151 )
152 # Chi2 p-value calculation
153 chi_p_val = 1 - chi2.cdf(chi_sqr, chi_dof)
154
155 # Results print out
156 # Gyro fitter result title
157 print()
158 print(f"'Gyro fitter results ' + title:<20}")
159 print("=" * 30)
160 # Print fit parameters
161 print(f"'Gyro fitter fitted parameters':<15}")
162 print(f"'A:':<20>{params[0]:>10.5g}")
163 print(f"'B:':<20>{params[1]:>10.5g}")
164 print(f"'a:':<20>{params[2]:>10.5g}")
165 print(f"'b:':<20>{params[3]:>10.5g}")
166 print(f"'Low freq slope:':<20>{params[2]:>10.5g}")
167 print(f"'High freq slope:':<20>{params[2]-params[3]:>10.5g}")
168 print()
169 # Print chi2 results
170 print(f"'Chi-square test result':<20}")
171 print(f"'Chi-square:':<20>{chi_sqr:>10.5g}")
172 print(f"'p-value:':<20>{chi_p_val:>10.5g}")

```

```

173     print("=" * 30)
174     print()
175
176     # Function return
177     return (params, cov, chi_sqr, chi_p_val)
178
179
180 # %% Plas fitter
181 # Plas fitter function
182 def plas_fitter(data_x, data_y, cut):
183     """
184     Parameters
185     -----
186     data_x : array
187         Combined freq data array.
188     data_y : array
189         Combined flux data array.
190     cut : float
191         Cut-off point for different fits.
192
193     Returns
194     -----
195     params : array
196         Fit parameters array.
197     cov : float
198         Covariance matrix of the fit.
199     chi_sqr :float
200         Chi-squared value of the fit.
201     chi_p_val : float
202         The p-value from the chi-squared test.
203     """
204     # Generate filtered data
205     data_x, data_y = (
206         data_x[data_x < cut],
207         data_y[data_x < cut],
208     )
209
210     # Initial parameter guess
211     param_guess = [1, 1] # param_c, param_k
212
213     # Curve fit results
214     params, cov = curve_fit(plas_model, data_x, data_y, p0=param_guess)
215
216     # Residuals generator
217     # Get fitted model
218     fit_model = plas_model(data_x, *params)
219     # Residual generator
220     fit_resid = data_y - fit_model
221
222     # Chi2 Tester
223     # Chi2 calculation and dof generation
224     chi_sqr, chi_dof = (
225         np.sum((fit_resid) ** 2 / fit_model),
226         len(data_x) - len(params),
227     )
228     # Chi2 p-value calculation
229     chi_p_val = 1 - chi2.cdf(chi_sqr, chi_dof)
230

```



```

231 # Results print out
232 # Gyro fitter result title
233 print()
234 print(f"{'Plas fitter results':<20}")
235 print("=" * 30)
236 # Print fit parameters
237 print(f"{'Plas fitter fitted parameters':<20}")
238 print(f"{'c':<20}{params[0]:>10.5g}")
239 print(f"{'k':<20}{params[1]:>10.5g}")
240 print()
241 # Print chi2 results
242 print(f"{'Chi-square test result':<20}")
243 print(f"{'Chi-square':<20}{chi_sqr:>10.5g}")
244 print(f"{'p-value':<20}{chi_p_val:>10.5g}")
245 print("=" * 30)
246 print()
247
248 # Function return
249 return (params, cov, chi_sqr, chi_p_val)
250
251
252 # %% Gyro model denoiser (cut plas model)
253 def gyro_pass(data_freq, data_flux, cut, plas_param):
254     """
255     Parameters
256     -----
257     data_freq : array
258         Peak time freq array.
259     data_flux : array
260         Peak time flux array.
261     cut : float
262         Low freq cut-off value.
263     plas_param : tuple
264         Tuple of plas model fit param.
265
266     Returns
267     -----
268     data_y_gyro : array
269         Denoised flux array for gyro fitting.
270     """
271     # Cut-off array local repo
272     data_x, data_y, data_y_base = (
273         data_freq[data_freq < cut],
274         data_flux[data_freq < cut],
275         data_flux[data_freq >= cut],
276     )
277
278     # Get denoised flux array in plas model domain
279     data_y_pass = data_y - plas_model(data_x, *plas_param)
280
281     # Get denoised flux data array for gyro fitting
282     data_y_gyro = np.concatenate([data_y_pass, data_y_base])
283
284     # Return denoised result
285     return data_y_gyro

```

## F. Code file: data\_plotter.py

```

1  """
2  This is the data plotter script of the radio data analysis project.
3
4  Created on Wed Mar 15 2023
5
6  @author: Yang-Taotao
7  """
8  # %% Library import
9  # Library import
10 import numpy as np
11 import matplotlib.pyplot as plt
12 import scienceplots
13
14 # Custom module import
15 from data_fitter import gyro_model, plas_model, fit_label
16
17 # %% Plot style config
18 # Plot style configuration
19 plt.style.use(["science", "notebook", "grid"])
20
21 # %% NoRP plotter
22 # NoRP log log plotter - flux vs freq at each time - 100 index = 10 s
23 def norp_plotter(arg):
24     """
25     Parameters
26     -----
27     arg : tuple
28         Plotter argument parameters.
29
30     Returns
31     -----
32     None.
33     """
34     # Local variable repo
35     (
36         data_norp_tim_valid,
37         data_norp_fi_peak,
38         data_norp_freq,
39         data_norp_peak_time,
40     ) = [arg[i] for i in range(len(arg))]
41
42     # Plot range limiter
43     # Gain peak value time array index
44     peak = (
45         np.where(data_norp_tim_valid == data_norp_peak_time)[0][0], # peak
46         300, # peak_gap
47         150, # gap
48     )
49     # Plot data range limiter at +- 30s --> peak_gap >= 300
50     # Structure - (peak_start, peak_end)
51     peak_idx = (
52         max(0, peak[0] - peak[1]), # peak_start
53         min(data_norp_fi_peak.shape[0], peak[0] + peak[1]), # peak_end
54     )
55
56     # Plot with loops

```

```

57 plt_0 = [
58     plt.plot(
59         data_norp_freq,
60         np.mean(
61             data_norp_fi_peak[i : i + peak[2]], axis=0
62         ), # Mean data calculator
63         "+-",
64         markersize=10,
65         label="NoRP " + data_norp_tim_valid[i],
66     )
67     for i in range(peak_idx[0], peak_idx[1], peak[2])
68 ]
69
70 # Plot axis scale definer
71 plt.xscale("log")
72 plt.yscale("log")
73
74 # Plot customizations
75 plt.xlabel("NoRP frequencies (GHz)", fontsize=14)
76 plt.ylabel("NoRP flux (SFU)", fontsize=14)
77 plt.legend(fontsize=10)
78
79 # Save and close
80 plt.savefig("./media/figure_01_norp.png")
81 plt.close()
82
83 # Return function call
84 return plt_0
85
86
87 # %% RSTN plotter
88 # RSTN log log plotter - flux vs freq at each time - 200 index = 190 s
89 def rstn_plotter(arg):
90     """
91     Parameters
92     -----
93     arg : tuple
94         Plotter argument parameters.
95
96     Returns
97     -----
98     None.
99     """
100     # Local variable repo
101     (
102         data_apl_tim,
103         data_phf_tim,
104         data_apl_fi_peak,
105         data_phf_fi_peak,
106         data_apl_freq,
107         data_phf_freq,
108         data_norp_peak_time,
109     ) = [arg[i] for i in range(len(arg))]
110
111     # Plot range limiter
112     # Gain peak value time array index
113     peak = (
114         np.where(data_apl_tim == data_norp_peak_time)[0][0], # peak_apl

```

```

115     np.where(data_phf_tim == data_norp_peak_time)[0][0], # peak_phf
116     60, # peak_gap
117     60, # gap
118 )
119 # Plot data range limiter at +- 30s
120 peak_idx = (
121     max(0, peak[0] - peak[2]), # peak_apl_start
122     min(data_apl_fi_peak.shape[0], peak[0] + peak[2]), # peak_apl_end
123     max(0, peak[1] - peak[2]), # peak_phf_start
124     min(data_phf_fi_peak.shape[0], peak[1] + peak[2]), # peak_phf_end
125 )
126
127 # Plot generation
128 # Plot apl with loops
129 plt_0 = [
130     plt.plot(
131         data_apl_freq,
132         np.mean(
133             data_apl_fi_peak[i : i + peak[3]], axis=0
134         ), # Mean data calculator
135         "x-",
136         markersize=10,
137         label="RSTN_apl " + data_apl_tim[i],
138     )
139     for i in range(peak_idx[0], peak_idx[1], peak[3])
140 ]
141 # Plot phf with loops
142 plt_1 = [
143     plt.plot(
144         data_phf_freq,
145         np.mean(
146             data_phf_fi_peak[i : i + peak[3]], axis=0
147         ), # Mean data calculator
148         "o-",
149         markerfacecolor="none",
150         markersize=10,
151         label="RSTN_phf " + data_phf_tim[i],
152     )
153     for i in range(peak_idx[2], peak_idx[3], peak[3])
154 ]
155
156 # Plot axis scale definer
157 plt.xscale("log")
158 plt.yscale("log")
159
160 # Plot customizations
161 plt.xlabel("RSTN frequencies (GHz)", fontsize=14)
162 plt.ylabel("RSTN flux (SFU)", fontsize=14)
163 plt.legend(fontsize=10)
164
165 # Save and close
166 plt.savefig("./media/figure_02_rstn.png")
167 plt.close()
168
169 # Return function call
170 return (plt_0, plt_1)
171
172

```

```

173 # %% Combined plotter
174 # Combined plotter
175 def log_plotter(arg):
176     """
177     Parameters
178     -----
179     arg : tuple
180         Plotter argument parameters.
181
182     Returns
183     -----
184     None.
185     """
186     # Local variable repo
187     (
188         data_norp_tim_valid,
189         data_apl_tim,
190         data_phf_tim,
191         data_norp_fi_peak,
192         data_apl_fi_peak,
193         data_phf_fi_peak,
194         data_norp_freq,
195         data_apl_freq,
196         data_phf_freq,
197         data_norp_peak_time,
198     ) = [arg[i] for i in range(len(arg))]
199
200     # Plot range limiter
201     # Gain peak value time array index
202     peak = (
203         np.where(data_norp_tim_valid == data_norp_peak_time)[0][0], # peak
204         np.where(data_apl_tim == data_norp_peak_time)[0][0], # peak_apl
205         np.where(data_phf_tim == data_norp_peak_time)[0][0], # peak_phf
206         300, # peak_norp_gap
207         60, # peak_rstn_gap
208         150, # gap_norp
209         60, # gap_rstn
210     )
211     # Plot data range limiter at +- 30s
212     peak_idx = (
213         max(0, peak[0] - peak[3]), # peak_start
214         min(data_norp_fi_peak.shape[0], peak[0] + peak[3]), # peak_end
215         max(0, peak[1] - peak[4]), # peak_apl_start
216         min(data_apl_fi_peak.shape[0], peak[1] + peak[4]), # peak_apl_end
217         max(0, peak[2] - peak[4]), # peak_phf_start
218         min(data_phf_fi_peak.shape[0], peak[2] + peak[4]), # peak_phf_end
219     )
220
221     # Plot generation
222     # Plot norp with loops
223     plt_0 = [
224         plt.plot(
225             data_norp_freq,
226             np.mean(
227                 data_norp_fi_peak[i : i + peak[5]], axis=0
228             ), # Mean data calculator
229             "+-",
230             markersize=10,

```

```

231         label="NoRP " + data_norp_tim_valid[i],
232     )
233     for i in range(peak_idx[0], peak_idx[1], peak[5])
234 ]
235 # Plot apl with loops
236 plt_1 = [
237     plt.plot(
238         data_apl_freq,
239         np.mean(
240             data_apl_fi_peak[i : i + peak[6]], axis=0
241         ), # Mean data calculator
242         "x-",
243         markersize=10,
244         label="RSTN_apl " + data_apl_tim[i],
245     )
246     for i in range(peak_idx[2], peak_idx[3], peak[6])
247 ]
248 # Plot phf with loops
249 plt_2 = [
250     plt.plot(
251         data_phf_freq,
252         np.mean(
253             data_phf_fi_peak[i : i + peak[6]], axis=0
254         ), # Mean data calculator
255         "o-",
256         markerfacecolor="none",
257         markersize=10,
258         label="RSTN_phf " + data_phf_tim[i],
259     )
260     for i in range(peak_idx[4], peak_idx[5], peak[6])
261 ]
262
263 # Plot axis scale definer
264 plt.xscale("log")
265 plt.yscale("log")
266
267 # Plot customizations
268 plt.xlabel("NoRP and RSTN frequencies (GHz)", fontsize=14)
269 plt.ylabel("NoRP and RSTN flux (SFU)", fontsize=14)
270 plt.legend(fontsize=10)
271
272 # Save and close
273 plt.savefig("./media/figure_03_combined.png")
274 plt.close()
275
276 # Return function call
277 return (plt_0, plt_1, plt_2)
278
279
280 # %% Combined peak time plotter
281 # Combined peak time plotter
282 def log_avg_plotter(arg):
283     """
284     Parameters
285     -----
286     arg : tuple
287         Plotter argument parameters.
288 
```



```

289 Returns
290 -----
291 None.
292 """
293 # Local variable repo
294 (
295     data_norp_tim_valid,
296     data_apl_tim,
297     data_phf_tim,
298     data_norp_fi_peak,
299     data_apl_fi_peak,
300     data_phf_fi_peak,
301     data_norp_freq,
302     data_apl_freq,
303     data_phf_freq,
304     data_norp_peak_time,
305 ) = [arg[i] for i in range(len(arg))]
306
307 # Plot range limiter
308 # Gain peak value time array index
309 peak = (
310     np.where(data_norp_tim_valid == data_norp_peak_time)[0][0], # peak
311     np.where(data_apl_tim == data_norp_peak_time)[0][0], # peak_apl
312     np.where(data_phf_tim == data_norp_peak_time)[0][0], # peak_phf
313     300, # peak_norp_gap
314     60, # peak_rstn_gap
315     150, # gap_norp
316     60, # gap_rstn
317 )
318 # Plot data range limiter at +- 30s
319 peak_idx = (
320     max(0, peak[0] - peak[3]), # peak_start
321     min(data_norp_fi_peak.shape[0], peak[0] + peak[3]), # peak_end
322     max(0, peak[1] - peak[4]), # peak_apl_start
323     min(data_apl_fi_peak.shape[0], peak[1] + peak[4]), # peak_apl_end
324     max(0, peak[2] - peak[4]), # peak_phf_start
325     min(data_phf_fi_peak.shape[0], peak[2] + peak[4]), # peak_phf_end
326 )
327
328 # Generate peak time averaged data array
329 data_norp_peak_avg, data_apl_peak_avg, data_phf_peak_avg = (
330     data_norp_fi_peak[peak_idx[0] : peak_idx[1], :],
331     data_apl_fi_peak[peak_idx[2] : peak_idx[3], :],
332     data_phf_fi_peak[peak_idx[4] : peak_idx[5], :],
333 )
334
335 # Plot generation
336 # Plot norp with loops
337 plt.plot(
338     data_norp_freq,
339     np.mean(data_norp_peak_avg, axis=0),
340     "+-",
341     markersize=10,
342     label="NoRP " + data_norp_tim_valid[peak[0]],
343 )
344 # Plot apl with loops
345 plt.plot(
346     data_apl_freq,

```

```

347     np.mean(data_apl_peak_avg, axis=0),
348     "x-",
349     markersize=10,
350     label="RSTN_apl " + data_apl_tim[peak[1]],
351 )
352 # Plot phf with loops
353 plt.plot(
354     data_phf_freq,
355     np.mean(data_phf_peak_avg, axis=0),
356     "o-",
357     markerfacecolor="none",
358     markersize=10,
359     label="RSTN_phf " + data_phf_tim[peak[2]],
360 )
361
362 # Plot axis scale definer
363 plt.xscale("log")
364 plt.yscale("log")
365
366 # Plot customizations
367 plt.xlabel("Combined NoRP and RSTN frequencies (GHz)", fontsize=14)
368 plt.ylabel("Combined NoRP and RSTN flux (SFU)", fontsize=14)
369 plt.legend(fontsize=10)
370
371 # Save and close
372 plt.savefig("./media/figure_03_peak_average_combined.png")
373 plt.close()
374
375 # Return averaged flux array at peak time
376 return (data_norp_peak_avg, data_apl_peak_avg, data_phf_peak_avg)
377
378
379 # %% Peak plotter
380 # Peak time plotter
381 def peak_plotter(arg):
382     """
383     Parameters
384     -----
385     arg : tuple
386         Peak plotter argument with freq, flux, and peak time.
387
388     Returns
389     -----
390     None.
391
392     """
393     # Local variable repo
394     (
395         data_peak_freq,
396         data_peak_flux,
397         data_norp_peak_time,
398         gyro_param,
399         plas_param,
400         freq_cut,
401     ) = [arg[i] for i in range(len(arg))]
402
403     # Fitted function repo
404     data_gyro, data_plas = (

```

```

405     gyro_model(data_peak_freq, *gyro_param),
406     plas_model(data_peak_freq[data_peak_freq < freq_cut], *plas_param),
407 )
408
409 # Fit label local repo
410 label_gyro, label_plas = fit_label(gyro_param, plas_param)
411
412 # Plot generation
413 plt_0 = plt.plot(
414     data_peak_freq,
415     data_peak_flux,
416     ".",
417     color="black",
418     markersize=10,
419     markeredgecolor="black",
420     label="Flux sequence at peak time: " + data_norp_peak_time,
421 )
422 # Gyro fit curve - x >= 2
423 plt_1 = plt.plot(
424     data_peak_freq,
425     data_gyro,
426     "+--",
427     markersize=10,
428     markeredgecolor="red",
429     label=label_gyro,
430 )
431 # Plas fit curve - x < 2
432 plt_2 = plt.plot(
433     data_peak_freq[data_peak_freq < freq_cut],
434     data_plas,
435     "x--",
436     markersize=10,
437     markeredgecolor="blue",
438     label=label_plas,
439 )
440
441 # Plot axis scale definer
442 plt.xscale("log")
443 plt.yscale("log")
444
445 # Plot customizations
446 plt.xlabel("Combined NoRP and RSTN frequencies (GHz)", fontsize=14)
447 plt.ylabel("Combined NoRP and RSTN flux (SFU)", fontsize=14)
448 plt.legend(fontsize=10)
449 plt.savefig("./media/figure_03_peak_time.png")
450 plt.close()
451
452 # Return function call
453 return (plt_0, plt_1, plt_2)
454
455
456 # %% Denoised data peak time plotter
457 def denoise_plotter(arg):
458     # Local variable repo
459     (
460         data_peak_freq,
461         data_peak_flux_denoise,
462         data_norp_peak_time,

```

```

463     denoise_param,
464     plas_param,
465 ) = [arg[i] for i in range(len(arg))]
466
467 # Fitted gyro function repo
468 data_gyro = gyro_model(data_peak_freq, *denoise_param)
469
470 # Fit label local repo
471 label_gyro = fit_label(denoise_param, plas_param)[0]
472
473 # Plot generation
474 plt_0 = plt.plot(
475     data_peak_freq,
476     data_peak_flux_denoise,
477     ".",
478     color="black",
479     markersize=10,
480     label="Denoised flux sequence at peak time: " + data_norp_peak_time,
481 )
482 # Gyro fit curve
483 plt_1 = plt.plot(
484     data_peak_freq,
485     data_gyro,
486     "+--",
487     markersize=10,
488     markeredgecolor="red",
489     label=label_gyro,
490 )
491
492 # Plot axis scale definer
493 plt.xscale("log")
494 plt.yscale("log")
495
496 # Plot customizations
497 plt.xlabel(
498     "Combined NoRP and RSTN frequencies at peak time (GHz)", fontsize=14
499 )
500 plt.ylabel("Low frequency denoised flux (SFU)", fontsize=14)
501 plt.legend(fontsize=10)
502 plt.savefig("./media/figure_04_denoised.png")
503 plt.close()
504
505 # Return function call
506 return (plt_0, plt_1)
507
508
509 # %% Plot generator
510 # Define generator function
511 def plot_generator(arg):
512     """
513     Parameters
514     -----
515     arg : array
516         Plot generator argument with sub plotter arguments.
517
518     Returns
519     -----
520     results : None

```

```
521     A collection of plots.
522     """
523     # Local variable repo
524     arg_norp, arg_rstn, arg_combine, arg_peak, arg_denoise = [
525         arg[i] for i in range(len(arg))
526     ]
527
528     # Result compilation
529     results = (
530         norp_plotter(arg_norp),
531         rstn_plotter(arg_rstn),
532         log_plotter(arg_combine),
533         log_avg_plotter(arg_combine),
534         peak_plotter(arg_peak),
535         denoise_plotter(arg_denoise),
536     )
537
538     # Return function call
539     return results
```

## G. Code file: data\_legacy.py

```

1  """
2  This is the data plotter legacy script of the radio data analysis project.
3
4  Created on Tue Mar 28 2023
5
6  @author: Yang-Taotao
7  """
8  # Library import
9  import numpy as np
10 import matplotlib.pyplot as plt
11 import scienceplots
12
13 # Custom module import
14 from data_fitter import gyro_model, plas_model
15
16 # Plot style configuration
17 plt.style.use(["science", "notebook", "grid"])
18
19 # Plotters
20 # NORP plotter - flux vs time at each freq
21 def norp_plotter(
22     data_norp_tim_valid, data_norp_fi_peak, data_norp_peak_time, data_norp_freq
23 ):
24     # Plot with loops
25     [
26         plt.plot(
27             data_norp_tim_valid,
28             data_norp_fi_peak[:, i],
29             label=data_norp_freq[i],
30         )
31         for i in range(data_norp_fi_peak.shape[1])
32     ]
33
34     # Plotting for the peak value
35     plt.axvline(x=data_norp_peak_time, color="crimson", linestyle="--")
36
37     # Gain peak value x-axis index
38     peak = np.where(data_norp_tim_valid == data_norp_peak_time)[0][0]
39     x_lim_left, x_lim_right = (
40         data_norp_tim_valid[peak - 2000],
41         data_norp_tim_valid[peak + 2000],
42     )
43
44     # Ticks declutter
45     plt.xticks(data_norp_tim_valid[::500], rotation=90, fontsize=10)
46
47     # Plot range limiter
48     plt.xlim(x_lim_left, x_lim_right)
49     plt.ylim(bottom=0)
50
51     # Plot customizations
52     plt.xlabel("Time", fontsize=14)
53     plt.ylabel("Valid NoRP flux negating quiet sun", fontsize=14)
54     plt.title("NoRP quiet sun subtracted flux against time", fontsize=16)
55     plt.legend(fontsize=10)
56     plt.show()

```

```

57
58
59 # RSTN plotter - preliminary
60 def rstn_plotter(
61     data_apl_tim,
62     data_phf_tim,
63     data_apl_flux_peak,
64     data_phf_flux_peak,
65     data_apl_freq,
66     data_phf_freq,
67 ):
68     # Plot apl data with loops
69     [
70         plt.plot(
71             data_apl_tim, data_apl_flux_peak[:, i], label=data_apl_freq[i]
72         )
73         for i in range(data_apl_flux_peak.shape[1])
74     ]
75     # Plot phf data with loops
76     [
77         plt.plot(
78             data_phf_tim, data_phf_flux_peak[:, i], label=data_phf_freq[i]
79         )
80         for i in range(data_phf_flux_peak.shape[1])
81     ]
82
83     # Ticks declutter
84     plt.xticks(data_apl_tim[::2000], rotation=90, fontsize=10)
85
86     # Plot range limiter
87     plt.ylim(bottom=0)
88
89     # Plot customizations
90     plt.xlabel("Time", fontsize=14)
91     plt.ylabel("Valid RSTN flux negating quiet sun", fontsize=14)
92     plt.title("RSTN quiet sun subtracted flux again time", fontsize=16)
93     plt.legend(fontsize=10)
94     plt.show()
95
96
97 # Combined plotter - preliminary
98 def combined_plotter(
99     data_norp_tim_valid,
100    data_norp_fi_peak,
101    data_norp_peak_time,
102    data_apl_tim,
103    data_phf_tim,
104    data_apl_flux_peak,
105    data_phf_flux_peak,
106 ):
107     # Plot with loops
108     [
109         plt.plot(data_norp_tim_valid, data_norp_fi_peak[:, i])
110         for i in range(data_norp_fi_peak.shape[1])
111     ]
112     [
113         plt.plot(data_apl_tim, data_apl_flux_peak[:, i])
114         for i in range(data_apl_flux_peak.shape[1])

```

```

115 ]
116 [
117     plt.plot(data_phf_tim, data_phf_flux_peak[:, i])
118     for i in range(data_phf_flux_peak.shape[1])
119 ]
120
121 # Plotting for the peak value
122 plt.axvline(x=data_norp_peak_time, color="crimson", linestyle="--")
123
124 # Plot range limiter
125 plt.ylim(bottom=0)
126
127 # Plot customizations
128 plt.xlabel("Time", fontsize=14)
129 plt.ylabel("Valid flux negating quiet sun", fontsize=14)
130 plt.title("Quiet sun subtracted flux again time", fontsize=16)
131 plt.show()
132
133
134 # %% Sepearate combined fit plotter
135 # Define fit plotter function
136 def fit_plotter(arg):
137     # Local variable repo
138     (
139         data_norp_freq,
140         data_apl_freq,
141         data_phf_freq,
142         data_peak_flux_norp,
143         data_peak_flux_apl,
144         data_peak_flux_phf,
145         norp_gyro_param,
146         norp_plas_param,
147         apl_gyro_param,
148         apl_plas_param,
149         phf_gyro_param,
150         phf_plas_param,
151     ) = [arg[i] for i in range(len(arg))]
152
153 # Plot generation
154 plt.plot(
155     data_norp_freq,
156     data_peak_flux_norp,
157 )
158 plt.plot(
159     data_apl_freq,
160     data_peak_flux_apl,
161 )
162 plt.plot(
163     data_phf_freq,
164     data_peak_flux_phf,
165 )
166 plt.plot(
167     data_norp_freq,
168     gyro_model(data_norp_freq, *norp_gyro_param),
169 )
170 plt.plot(
171     data_norp_freq,
172     plas_model(data_norp_freq, *norp_plas_param),

```



```
173 )
174 plt.plot(
175     data_apl_freq,
176     gyro_model(data_apl_freq, *apl_gyro_param),
177 )
178 plt.plot(
179     data_apl_freq,
180     plas_model(data_apl_freq, *apl_plas_param),
181 )
182 plt.plot(
183     data_phf_freq,
184     gyro_model(data_phf_freq, *phf_gyro_param),
185 )
186 plt.plot(
187     data_phf_freq,
188     plas_model(data_phf_freq, *phf_plas_param),
189 )
190
191 # Plot axis scale definer
192 plt.xscale("log")
193 plt.yscale("log")
194
195 # Plot customizations
196 plt.xlabel("Frequencies at peak time (GHz)", fontsize=14)
197 plt.ylabel("Valid quiet sun filtered flux", fontsize=14)
198 plt.title("Quiet sun flux at peak time", fontsize=16)
199 plt.legend(fontsize=10)
200 plt.savefig("./media/figure_fit.png")
201 plt.close()
```