MCMC Fitting Code Guide

Henry Davenport

September 12, 2021

Contents

1 Code Overview **1**
1.1 Fit Equation . . . 1
1.2 Sampler . . . 2

2 Code Requirements **2**
2.1 EMCEE Download . . . 3
2.2 Corner Download . . . 3
2.3 Files Required . . . 3

3 Setting Up the Code **3**
3.1 Importing the code . . . 3
3.2 Choosing the Parameters to be Fitted . . . 3
3.3 Speeding Up the Fitting . . . 4
3.4 Number of Samples and Burn-In Times . . . 4
3.5 Fitting Data with Low Duty Cycle . . . 4
3.6 Mode Visibilities and m-component Height Ratios . . . 5

4 Important Functions **5**
4.1 Importing and Fourier-Transforming the Data (high duty cycle) . . . 5
4.2 Importing and Fourier-Transforming the Data (low duty cycle) . . . 5
4.3 Run All Peak Fits . . . 6

5 Code Examples **7**
5.1 GOLF Fit . . . 7
5.2 BiSON Fit . . . 7

6 Code Output **8**
6.1 fit_parameters.txt . . . 8
6.2 Corner plots . . . 8
6.3 Fit plots . . . 8
6.4 autocorrelation_times.txt . . . 8

7 Troubleshooting **9**
7.1 Invalid Value in double_scalars . . . 9
7.2 Likelihood function returning Nan . . . 9
7.3 Running Code on Nenneke . . . 9

1 Code Overview

1.1 Fit Equation

The power spectrum for a given time series is calculated using the Fourier transform of the velocity or intensity data. The $l = 0$, 2, and $l = 1$, 3 mode pairs are fitted simultaneously due to their proximity.

Each mode component, defined by radial order n, angular degree l, and azimuthal order m, is fitted with an asymmetric Lorentzian ([1]) of the form:

$$P(x) = H\frac{(1 + Bx)^2 + B^2}{1 + x^2}, \tag{1}$$

where the peak height is H and $x = 2(\nu - \nu_0)/\Delta\nu$ with central frequency ν_0 and linewidth $\Delta\nu$. The degree of asymmetry is given by B such that when $B = 0$ the $P(x)$ profile is a symmetric Lorentzian. An additional flat offset is fitted to account for uncorrelated background noise. generally it is best to fix the ratio of the heights of the m-split multiplets in the l=2 and l=3 modes. See [2] for values to use. These are different for different instruments.

The program allows a range of different parameters to be kept constant or varied. For example, at high frequencies the large linewidths mean that the l=0,2 and l=1,3 modes overlap considerably. Therefore, it is often best to fix the splitting parameter (which describes the frequency shift between the m components of the mode). For modes with eigenfrequencies less than 3400μHz the following parameters are often best to fit:

1. An eigenfrequency for each mode.
2. One linewidth per mode. Each m mode component was assumed to have the same width.
3. One asymmetry per pair of modes.
4. One rotational frequency splitting parameter per mode (apart from for the $l = 0$ peak which is not split).
5. A single peak height for each mode.
6. A constant flat background offset across the whole fitting window.

For modes with eigenfrequencies above 3400μHz the modes in each $l = 0, 2$ and $l = 1, 3$ pair because of the increasing overlap, it is often best to fix the frequency splitting of the m components in order to improve the fit quality. This splitting can be fixed at the average of the splittings found below 3400μHz. Additionally, it is recommended that both modes in the fitting window are fitted with the same width parameter.

1.2 Sampler

The parameters are fitted in the code by using the EMCEE sampler to sample the likelihood function. The likelihood function is given by a χ^2 with 2 degrees of freedom distribution because this is distribution of the noise in the power spectra. Emcee, is used to sample the likelihood function and find the marginal distributions of the fit parameters. The distribution of the natural logarithms of the widths and peak heights are found because these have quasi-normal marginal distributions whereas the straightforward parameter values give a skewed distribution. The final parameter values of a fit are given by the sample with the highest likelihood and the upper and lower errors for each parameter are found using the 16th and 84th percentiles of the marginal distributions. This is one standard deviation for a symmetric distribution but allows for asymmetric distributions.

2 Code Requirements

The code is dependent on a number of libraries. The standard ones:

- Numpy
- Matplotlib
- Astropy
- Scipy

The non-standard ones are:

- EMCEE
- Corner

2.1 EMCEE Download

Can download using pip:

```
python -m pip install -U pip
pip install -U setuptools setuptools_scm pep517
pip install -U emcee
```

Or can alternatively use conda if using Anaconda package manager:

```
conda update conda
conda install -c conda-forge emcee
```

See the EMCEE Documentation for further information.

2.2 Corner Download

Download using pip:

```
python -m pip install corner
```

or conda:

```
conda install -c astropy corner
```

2.3 Files Required

Place the following files all in the same directory:

- mcmc.py file – this contains the fit code
- Priors.txt file - used for the priors.
- A ".fits" format file to be fitted by the code.

3 Setting Up the Code

3.1 Importing the code

To use the code you can either create a new python file which imports the functions from mcmc.py or directly add your script to the end of the mcmc.py file. After importing you will need to first create an instance of the MCMC class as in the example below:

```python
from mcmc import MCMC
mcmc = MCMC()
# insert rest of code here
```

3.2 Choosing the Parameters to be Fitted

First, if you want the Boolean flags shown in Table 3.2 to be different from their default values you can do so as in the following example:

```python
mcmc.Different_Widths = True
```

Keeping extra parameters constant can be achieved by adding the parameter name (as a string) to the parameters_kept_constant variable. This means this value will be kept constant at the value in the prior file. For example:

```python
mcmc.parameters_kept_constant = ["scale", "splitting"]
```

The parameters that can be kept constant are:

```python
"freq", "splitting", "width", "power", "asymmetry", "background", "scale"
```

Number	Default Value	Description
Different_Widths	False	If true, then both peaks in the fitting window are fitted with different width parameters. Else, if false then they are fitted with the same width parameter.
Fit_Background	True	If true, then fits a constant offset background across the whole fit window. If false, then no background is fitted.
Fit_Asymmetric	True	If true, then the modes are fitted with asymmetric Lorentzian curves. If, false then they are fitted with symmetric Lorentzian curves.
Different_Asymmetries	False	If true, then both peaks in the fitting window are fitted with different asymmetry parameters. If false, then they share an asymmetry parameter in the fit.

Table 1: Boolean flags and their functions

3.3 Speeding Up the Fitting

Generally if the computer you are running the code on has multiple cores then the fitting can be sped up substantially by running multiple peak fits at once as different processes.

To do this set the number_of_processes variable to the number of processes you want to run. On Nenneke it is typically quickest to run around or over 10 processes at once though this depends on how intensive each process is (e.g. shorter time series can generally be run with more processes). so this requires some experimentation. On a normal Laptop run one less process than the number of cores on the CPU for quickest fitting and to allow other tasks to run. Example

```
mcmc.number_of_processes = 3
```

3.4 Number of Samples and Burn-In Times

Recommended Burn-In: 1000
Recommended Number of Samples: 10,000

The walkers all start in a tight cluster around the initial guess. They then move from these initial positions in a semi-random walk and explore the likelihood surface. The first samples are therefore highly correlated and biased toward the starting location. We do not want these samples in our final samples so a burn-in is required. This runs the sampler for a number of iterations, but we then discard these samples in our final analysis, they serve only to ensure that all the samples in our final sampler run are uncorrelated with the starting location. For more on Burn-In periods read: `https://emcee.readthedocs.io/en/stable/tutorials/quickstart/?highlight=burn#how-to-sample-a-multi-dimensional-gaussian`

The length of the burn-in period required cannot be calculated, and it is also not possible to prove that a chain is sufficiently converged. However, in practice you can tell a chain hasn't converged from its corner plot. Tight clusters of samples away from the main Gaussian peak often imply a longer burn-in is required. The autocorrelation time of a chain can be used to check to see if the chain is long enough (it should be roughly 50 times this length) and the code runs extra iterations if they are required.

In addition the acceptance fractions are an important diagnostic. These give the fraction of the proposed walker position changes that are accepted. These should be generally above 0.3. Sometimes one walker gets trapped in a region where it remains for a long time leading to acceptance fractions under 0.01. The code checks the acceptance fractions and re-runs the sampling if they are too low. It returns the message "Some Chains have very low acceptance fractions so rerun." if this occurs.
To change the burn-in and number of samples:

```
mcmc.burnin = 10000
mcmc.no_samples = 20000
```

3.5 Fitting Data with Low Duty Cycle

If the time series has a lots of gaps (e.g. it is ground based like BiSON) then the window function will need to be converged with the model at each iteration. To do this follow the code example for BiSON.

3.6 Mode Visibilities and m-component Height Ratios

The visibilities of the modes and the relative heights of different m components of each l peak are different for different instruments. They need to be set for each different instrument, for GOLF:

```
# the ratios of the heights of the m split components of the l=2 and l=3 modes
mcmc.l2_m_scale = 0.634
mcmc.l3_m_scale = 0.400
# the mode visibilities
mcmc.l1_visibility = 1.505
mcmc.l2_visibility = 0.62
mcmc.l3_visibility = 0.075
```

4 Important Functions

The following functions are those that you need to use in order to run the peak-bagging.

4.1 Importing and Fourier-Transforming the Data (high duty cycle)

This is all done by the function:

```
import_data(filename, start, end):
```

This takes as input:

- filename: the .fits filename of the data that you want to be fitted.
- start, end: These two parameters give the range of indices in the data that you want to Fourier transform and return. For example often you want to split the entire data set into one year segments and Fourier transform and then fit those individually.

This returns as output the power spectrum and corresponding frequency arrays in that order.

Example In order to return the second year of GOLF data in a .fits file the following code could be run:

```
# the number of data points in a complete year of GOLF data
# This is the number of seconds in a year divided by the cadence of the data
year_npts = 525600
# file to be fitted:
data_to_be_fitted = "GOLF_bw_rw_pm3_960411_200830_Sync_RW_dt60F_v1.fits"
power, freq = mcmc.import_data(data_to_be_fitted, 1 * year_npts, 2 * year_npts)
```

4.2 Importing and Fourier-Transforming the Data (low duty cycle)

For BiSON will want to fit by converging a window function with the model at each fitting iteration. Therefore import the data using the function:

```
import_data_window(filename, start, end)
```

This takes as input:

- filename: the .fits filename of the data that you want to be fitted.
- start, end: These two parameters give the range of indices in the data that you want to Fourier transform and return. For example often you want to split the entire data set into one year segments and Fourier transform and then fit those individually.

This returns as output the power spectrum, corresponding frequency array and the Fourier transformed window function in that order.

Example In order to return the second year of GOLF data in a .fits file the following code could be run:

```
# the number of data points in a complete year of BiSON data
# This is the number of seconds in a year divided by the cadence of the data
year_npts = 788400
# file to be fitted:
data_to_be_fitted = "bison-allsites-alldata-waverage-fill.fits"
power, freq, Pow_Win = mcmc.import_data_window(data_to_be_fitted, 5 * year_npts, 6 *
↪   year_npts)
```

4.3 Run All Peak Fits

The function fit_all_peaks fits all the peaks in a specified frequency range.

```
fit_all_peaks(power, freq, foldername, freq_range=(2000, 3500),
↪   pars_constant=parameters_kept_constant, splitting=0.40, modes_per_fit = 1,
↪   window_width = 25, pow_win = None)
```

The inputs are:

- power: the power spectrum array (from import_data)
- freq: the frequency array (from import_data)
- foldername: the folder name for all the output
- freq_range: tuple with range of frequencies that you want to fit over in μHz. If no value given then the default is to fit over range 2000μHz to 3500μHz.
- pars_constant: this is a list of the parameters that are to be kept constant, if no input is given the global variable parameters_kept_constant is used.
- splitting: this gives the initial value of the mode splitting. Default is 0.40.
- modes_per_fit: this value sets the number of peak pairs (e.g. l=1,3 or l=2,0) in each fit window. For example if set to 3 then the l=1,3 peaks either side of a l=0,2 peak would be fitted simultaneously.
- window_width: the fitting window is set to be from the initial guess frequency of the lowest frequency mode minus the window_width (in μHz) and to the initial guess frequency of the largest frequency mode plus this width. Default is 40μHz.
- pow_win: if using BiSON or other data with low duty cycle then pass the 3rd output of the Pow_Win output of the mcmc.import_data_window to the function using this.

Example The following code shows how you can fit different frequency ranges with different parameters kept constant using the fit_all_peaks function:

```
from mcmc import MCMC
mcmc = MCMC()







year_npts = 525600
power, freq = mcmc.import_data(data_to_be_fitted, 0, year_npts)
folder_name = mcmc.create_directory()
# first keep just scale constant
mcmc.parameters_kept_constant = ["scale"]
average_splitting = mcmc.fit_all_peaks(power, freq, folder_name,
                pars_constant=mcmc.parameters_kept_constant,
                freq_range=(2500, 3400))
# for high frequencies want the splitting also kept constant
# at the average value from lower frequencies
mcmc.parameters_kept_constant = ["scale", "splitting"]
mcmc.fit_all_peaks(power, freq, folder_name, pars_constant=parameters_kept_constant,
                freq_range=(3400, 4000), splitting = average_splitting)
```

5 Code Examples

5.1 GOLF Fit

The following code cycles through all GOLF data with fits of 365 day time series with a four time shift.

```
from mcmc import MCMC
mcmc = MCMC()
filename = "GOLF_bw_rw_pm3_960411_200830_Sync_RW_dt60F_v1.fits"
mcmc.number_of_processes = 10

# the ratios of the heights of the m split components of the l=2 and l=3 modes
mcmc.l2_m_scale = 0.634
mcmc.l3_m_scale = 0.400
# the mode visibilities
mcmc.l1_visibility = 1.505
mcmc.l2_visibility = 0.62
mcmc.l3_visibility = 0.075


# This is the number of seconds in a year divided by the cadence
# Total number of measurements in a year
year_npts = 525600
# length of each fit in years: (i.e. one year)
no_years_fit = 1.0
# number of measurements in each fit:
length_of_fit = year_npts * no_years_fit
# total number of measurements in the whole file:
total_npts = 12827520
number_of_fits = math.floor(total_npts/length_of_fit)
shift = 0.25 * year_npts
for i in range(number_of_fits):
    power, freq = mcmc.import_data(filename, i*shift, i*shift + length_of_fit)
    folder_name = mcmc.create_directory()
    # first run low frequency (2000 to 3300 micro-Hz) peaks with just the scale
    ↪   parameter constant
    mcmc.parameters_kept_constant = ["scale"]
    average_splitting = mcmc.fit_all_peaks(power, freq, folder_name,
    ↪   pars_constant=mcmc.parameters_kept_constant, freq_range=(2000, 3300),
    ↪   modes_per_fit=3)
    # Run high frequency peaks with scale and splitting constant.
    # splitting is set to average of splittings for all low frequency fits.
    parameters_kept_constant = ["scale", "splitting"]
    mcmc.fit_all_peaks(power, freq, folder_name,
    ↪   pars_constant=mcmc.parameters_kept_constant, freq_range=(3400, 4000),
    ↪   splitting = average_splitting, modes_per_fit=3)
```

5.2 BiSON Fit

In addition for the BiSON fit the window function is converged with the model of the power spectrum at each iteration. To run 365 day fits over the whole data set with a four time shift use the following code:

```
from mcmc import MCMC
mcmc = MCMC()
mcmc.number_of_processes = 10
data_to_be_fitted = "bison-allsites-alldata-waverage-fill.fits"
# This is the number of seconds in a year divided by the cadence
# Total number of measurements in a year (BiSON cadence is 40 seconds)
year_npts = 788400
```

```
# length of each fit in years: (i.e. one year)
no_years_fit = 1.0
# number of measurements in each fit:
length_of_fit = year_npts * no_years_fit
# total number of measurements in the whole file:
total_npts = 34909918
shift = math.floor(0.25 * year_npts)
number_of_fits = math.floor(total_npts / shift)
for i in range(number_of_fits):
    power, freq, Pow_Win = mcmc.import_data_window(data_to_be_fitted, i*shift, i*shift
    ↪   + length_of_fit)
    folder_name = mcmc.create_directory()
    # first run low frequency (2000 to 3300 micro-Hz) peaks with just the scale
    ↪   parameter constant

    mcmc.parameters_kept_constant = ["scale"]
    average_splitting = mcmc.fit_all_peaks(power, freq, folder_name,
    ↪   pars_constant=mcmc.parameters_kept_constant, freq_range=(3400, 4000),
    ↪   pow_win=Pow_Win)

    # Run high frequency peaks with scale and splitting constant.
    # splitting is set to average of splittings for all low frequency fits.

    parameters_kept_constant = ["scale", "splitting"]
    mcmc.fit_all_peaks(power, freq, folder_name,
    ↪   pars_constant=mcmc.parameters_kept_constant, freq_range=(3400, 4000),
    ↪   splitting = average_splitting, pow_win=Pow_Win)
```

6 Code Output

All the output per time series fitted is placed in a folder named “Output_” + the time at which the fit began. This folder is in the directory in which the code is. In the folder the following files can be found.

6.1 fit_parameters.txt

This file contains all the final fit parameters. The data is tab separated and the columns of the output are:

```
"l", "n", "freq", "freq error", "ln(power)", "ln(power) error", "splitting",
"splitting error", "scale", "scale error", "ln(linewidth)", "ln(linewidth) error",
"background", "background error", "asymmetry", "asymmetry error"
```

Important: Each error takes up two columns, the first of which is the lower error (the difference between the “best-fit” parameter and the 14th quartile) and the second is the upper error (the difference between the “best-fit” parameter and the 86th quartile).

6.2 Corner plots

These show the marginal distributions of the samples of each parameter. These are labelled with the l and n numbers of the modes that are fitted.

6.3 Fit plots

These show the output best fit of the MCMC code overlaid over the data that was fitted. Again the image is labelled with the l and n numbers of the modes in the fitting window.

6.4 autocorrelation_times.txt

This contains information about the quality of the fits. Generally this is only required for diagnostics and can be ignored. All the output from this is output to the terminal anyway.

7 Troubleshooting

7.1 Invalid Value in double_scalars

If you get the following error:

```
0%|          | 0/1000 [00:00<\?, ?it/s]
C:\Users\User\anaconda3\lib\site-packages\emcee\moves\red_blue.py:99:
RuntimeWarning: invalid value encountered in double_scalars
```

This means that the initial positions of some of the walkers is outside the range allowed by priors. To fix this: try making priors larger. Also check that the initial values input to the code are not negative if they shouldn't be: e.g. all parameters apart from asymmetry cannot be negative.

7.2 Likelihood function returning Nan

This error can be raised if the the fitting region isn't overlapping with the peaks you are trying to fit.

7.3 Running Code on Nenneke

Copy over the files required into a directory.
Open the terminal.
To get the libraries required:

```
module load Anaconda3/2019.03
```

Set your current directory to the directory with the code in:

```
cd /home/physics/_USER_ID_/Documents/_FOLDER_NAME_
```

Then run the file using python 3.

```
python3 Run.py
```

References

[1] R. Nigam and A. G. Kosovichev. Measuring the sun's eigenfrequencies from velocity and intensity helioseismic spectra: Asymmetrical line profile–fitting formula. *The Astrophysical Journal*, 505(1): L51–L54, sep 1998. doi:10.1086/311594. URL https://doi.org/10.1086/311594.

[2] Salabert, D., Ballot, J., and García, R. A. Mode visibilities in radial velocity and photometric sun-as-a-star helioseismic observations. *A&A*, 528:A25, 2011. doi:10.1051/0004-6361/201015946. URL https://doi.org/10.1051/0004-6361/201015946.