AFINO

Release 0.5

Nov 29, 2022

Contents

AFINO is a tool to enable searching for frequency enhancements or oscillations in time series data. AFINO was originally developed to study solar flare oscillations and has been adapted to be of general use.

AFINO is freely available, but should be treated as an experimental scientific tool. Below we provide code reference for the current package.

CHAPTER 1

Code Reference

1.1 afino_start

afino.afino_start.**analyse_series**(*times, flux, description=None, low_frequency_cutoff=None, savedir=None, overwrite_gauss_bounds=None, overwrite_extra_gauss_bounds=None, use_json=True, model_ids=[0, 1, 2]*)

Analyse a single, generic timeseries using the AFINO model comparison code.

Parameters

- **times** (*ndarray*) – an array of times
- **flux** (*ndarray*) – an array of data points
- **description** (*string, optional*) – a string descriptor of the analysis run, that is incorporated into output filenames
- **low_frequency_cutoff** (*float, optional*) – specifies a frequency above which the input Fourier spectrum is not analysed
- **savedir** (*string, optional*) – specifies a directory for output save files
- **use_json** (*bool*) – If set to True, saves analysis output in JSON format. If False, pickle format is used. Default is True.

afino.afino_start.**analyse_series_twobump**(*times, flux, description=None, low_frequency_cutoff=None, savedir=None, overwrite_gauss_bounds=None, overwrite_extra_gauss_bounds=None*)

Analyse a single, generic timeseries using the AFINO model comparison code., with extra models

afino.afino_start.**create_generic_summary_plot**(*ts, analysis_summary, description, low_frequency_cutoff=None, savedir=None*)

Create a summary plot showing the result of an AFINO analysis run.

afino.afino_start.**create_generic_summary_plot_twobump**(*ts*, *analysis_summary*, *description*, *low_frequency_cutoff=None*, *savedir=None*)

Create a summary plot showing the result of an AFINO analysis run.

1.2 afino_model_comparison

afino.afino_model_comparison.**model_comparison**(*ts*, *description=None*, *low_frequency_cutoff=None*, *overwrite_gauss_bounds=None*, *overwrite_extra_gauss_bounds=None*, *use_json=True*, *model_ids=[0, 1, 2]*)

Initiate the comparison of different models to the Fourier power spectrum of a timeseries.

1.3 afino_main_analysis

This routine initiates the AFINO analysis method for a particular chosen model. The fitting is done via SciPy, and the results, including best fit and associated BIC value, are stored in a dictionary and returned.

1.4 afino_model_fitting

afino.afino_model_fitting.**lnlike**(*variables*, *x*, *y*, *model_function*)

Log likelihood of the data given a model. Assumes that the data is exponentially distributed. Can be used to fit Fourier power spectra. :param variables: array like, variables used by model_function :param x: the independent variable (most often normalized frequency) :param y: the dependent variable (observed power spectrum) :param model_function: the model that we are using to fit the power spectrum :return: the log likelihood of the data given the model.

afino.afino_model_fitting.**prob_this_rchi2_or_larger**(*rchi2*, *m*, *nu*)

Parameters

- **rchi2** – reduced chi-squared value
- **m** – number of spectra considered
- **nu** – degrees of freedom

Returns

afino.afino_model_fitting.**rchi2**(*m*, *nu*, *rhoj*)

Goodness-of-fit estimator (Eq. 16)

Parameters

- **m** – number of spectra considered
- **nu** – degrees of freedom
- **rhoj** – sample to model ratio estimator

Returns A chi-square like goodness of fit estimator

Return type float

afino.afino_model_fitting.**rchi2distrib**(*m*, *nu*)
The distribution of rchi2 may be approximated by the analytical expression below. Comparing Eq. (2) with the implementation in scipy stats we find the following equivalencies: k (Nita parameter) = a (scipy stats parameter) theta (Nita parameter) = 1 / lambda (scipy stats value) :param m: number of spectra considered :param nu: degrees of freedom :return: a frozen scipy stats function that represents the distribution of the data

afino.afino_model_fitting.**rhoj**(*Sj*, *shatj*)
Sample to Model Ratio (SMR) estimator (Eq. 5)

Parameters

- **Sj** – random variables (i.e. data)
- **shatj** – best estimate of the model. Should be same length as Sj

Returns The Sample-to-Model ratio

Return type ndarray

1.5 afino_spectral_models

Power Spectrum Models

afino.afino_spectral_models.**bpow**(*a*, *f*)
Broken power law. This model assumes that there is a break in the power spectrum at some given frequency.

Parameters

- **a** (*ndarray[3]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index at frequencies lower than the break frequency
  - a[2] : break frequency
  - a[3] : the power law index at frequencies higher than the break frequency
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**bpow_const**(*a*, *f*)
Broken power law with constant. This model assumes that there is a break in the power spectrum at some given frequency. At high frequencies the power spectrum is dominated by the constant background.

Parameters

- **a** (*ndarray(5)*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index at frequencies lower than the break frequency
  - a[2] : break frequency
  - a[3] : the power law index at frequencies higher than the break frequency
  - a[4] : the natural logarithm of the constant background
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**broken_power_law_with_constant_with_lognormal**(*a*, *f*)
Broken power law with constant with lognormal. This model assumes that there is a break in the power spectrum

at some given frequency. At high frequencies the power spectrum is dominated by the constant background. At some particular frequency there is a lognormal (narrowband distribution)

Parameters

- **a** (*ndarray(5)*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index at frequencies lower than the break frequency
  - a[2] : break frequency
  - a[3] : the power law index at frequencies higher than the break frequency
  - a[4] : the natural logarithm of the constant background
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**constant**(*a*)
The power spectrum is a constant across all frequencies

Parameters **a** (*float*) – the natural logarithm of the power

afino.afino_spectral_models.**fnorm**(*f*, *normalization*)
Normalize the frequency spectrum.

afino.afino_spectral_models.**lognormal**(*a*, *f*)
A lognormal distribution

Parameters

- **a** (*ndarray(3)*) –
  - a[0] : the natural logarithm of the Gaussian amplitude
  - a[1] : the natural logarithm of the center of the Gaussian
  - a[2] : the width of the Gaussian in units of natural logarithm of the frequency
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**pow**(*a*, *f*)
Simple power law. This model assumes that the power spectrum is made up of a power law at all frequencies.

Parameters

- **a** (*ndarray[2]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**pow_const**(*a*, *f*)
Power law with a constant. This model assumes that the power spectrum is made up of a power law and a constant background. At high frequencies the power spectrum is dominated by the constant background.

Parameters

- **a** (*ndarray[2]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index
  - a[2] : the natural logarithm of the constant background

- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**pow_const_2gauss**(*a*, *f*)

Simple power law with a constant, plus a Gaussian bump, and a second Gaussian bump. The second bump is implemented to account for a persistent signal in the data, such as a spin period.

Parameters

- **a** (*ndarray[8]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index
  - a[2] : the natural logarithm of the constant background
  - a[3] : The amplitude of the first bump
  - a[4] : the frequency location of the first bump
  - a[5] : the width of the first bump
  - a[6] : The amplitude of the second bump
  - a[7] : the frequency location of the second bump
  - a[8] : the width of the second bump
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**pow_const_gauss**(*a*, *f*)

Simple power law with a constant, plus a Gaussian shaped bump. This model assumes that the powe spectrum is made up of a power law and a constant background. At high frequencies the power spectrum is dominated by the constant background.

Parameters

- **a** (*ndarray[2]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index
  - a[2] : the natural logarithm of the constant background
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**power_law_with_constant_with_lognormal**(*a*, *f*)

Power law with constant and a lognormal.

Parameters

- **a** (*ndarray[6]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the power law index
  - a[2] : the natural logarithm of the constant background
  - a[3] : the natural logarithm of the Gaussian amplitude
  - a[4] : the natural logarithm of the center of the Gaussian
  - a[5] : the width of the Gaussian in units of natural logarithm of the frequency
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**sum_of_pulses**(*a*, *f*)

Sum of pulses. This model is based on Aschwanden "Self Organized Criticality in Astrophysics", Eq. 4.8.23. Simulations implementing this equation come up with a shape that is modeled below.

Parameters

- **a** (*ndarray[3]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the scale frequency
  - a[2] : the power law index
- **f** (*ndarray*) – frequencies

afino.afino_spectral_models.**sum_of_pulses_with_constant**(*a*, *f*)

Sum of pulses plus constant. This model is based on Aschwanden "Self Organized Criticality in Astrophysics", Eq. 4.8.23, with a constant background to model detector noise.

Parameters

- **a** (*ndarray[3]*) –
  - a[0] : the natural logarithm of the normalization constant
  - a[1] : the scale frequency
  - a[2] : the power law index
  - a[3] : natural logarithm of the background constant
- **f** (*ndarray*) – frequencies

1.6 afino_utils

This module provides a number of convenience functions for use throughout the AFINO codebase.

class afino.afino_utils.**NumpyEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

This encoder converts numpy arrays to lists so that they can be saved in JSON format

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

afino.afino_utils.**model_string_from_id**(*id*)

This function converts an AFINO model ID integer to a string descriptor

afino.afino_utils.**relative_bics**(*saveresult*)
This convenience function calculates all the relative BIC comparisons from an AFINO save result.

Parameters **saveresult** (*dict*) – An AFINO save result dictionary. Can be restored from a previous save file

Returns **dbic_values** – A dictionary of relative BIC value comparisons, i.e. BICa - BICb for every a,b pair

Return type dict

afino.afino_utils.**restore_json_save_file**(*fname*)
This function restores an AFINO JSON save file and converts certain dictionary entries back into their original ndarray form. The output should be identical to that from the pickle save files

afino.afino_utils.**save_afino_results**(*results*, *use_json=False*, *description=None*)
This function saves the results of an AFINO analysis run to either a JSON file or a Pickle file.

Parameters

- **results** (*list*) – An AFINO results list, obtained via the model_comparison function
- **use_json** (*bool, optional*) – If True, use JSON format to save the analysis results. If False, use pickle format
- **description** (*string, optional*) – An optional descriptor for the result that is incorporated into the filename

1.7 afino_series

afino.**afino_series**
alias of *afino.afino_series*

A simple time series object

afino.afino_series.**prep_series**(*ts*)
put the data series into the form (I - <I>) / <I>. multiply the series by a Hanning window to aid the FFT process

CHAPTER 2

User Guide

This guide will introduce you to the basics of how to perform a time series analysis of AFINO. It will also walk through the output products and how to interpret the results.

2.1 Directory structure

By default, AFINO will save the results of an analysis in '~/afino_repository'. If this directory does not exist, it must be created. In this directory, two additional folders must be created. They are 'plots' and 'saves'.

The results can be saved in either JSON or Pickle format, via the `use_json` keyword. These will be placed in the 'saves' folder. AFINO will also produce a summary plot that will be placed in the 'plots' folder.

2.2 Running AFINO

The main executable function is `analyse_series` within the `afino_start` module. The example below shows a simple analysis of a sine wave in white noise:

```
import numpy as np
import afino
from afino import afino_start

tt = np.linspace(0,1000,1001)
flux = np.sin(0.1 * tt) + 3*np.random.random(1001)
afino_start.analyse_series(tt,flux, model_ids = [0,1,2,3], use_json = True)
```

Once the analysis is complete, AFINO will generate a results file (in this example a .json file) and a summary plot. The results will identify a strong peak at P ~ 63s as expected. The default locations for these results are mentioned above, but can be changed via the `savedir` keyword.

2.2.1 Additional options

There are some additional optional keywords that can be passed into the `analyse_series` function:

- *description*: a string descriptor of the analysis run that is incorporated into output filenames. If not set, a default naming construction is used.
- *low_frequency_cutoff*: specifies a frequency above which the input Fourier spectrum is not analysed.
- *overwrite_gauss_bounds*: None by default, this can be set to overwrite hardcoded parameter limits for model 1.

2.3 Inspecting Results

We can open a results file using the JSON Python module (or the pickle module if that option was selected.

```
import json
result = json.load(open('result = json.load(open(filename,'r'))
```

where `filename` corresponds to the appropriate save file. We can then inspect the contents of the file.

```
In [2]:  result.keys()
Out[2]: dict_keys(['m0', 'm1', 'm2', 'm3'])
```

Here we can see that the results file contains a dictionary. Each key 'm0','m1' corresponds to a fit of each chosen model. For example, if only models 0 and 1 were chosen to be fit in the analysis stage, then only keys 'm0' and 'm1' would be present. In this example, models 0, 1, 2, and 3 were all attempted, so their corresponding keys are present.

The results are a nested dictionary, so each key 'm0', 'm1' etc contains its own keys.

```
In [3]: result["m0"].keys()
Out[3]: dict_keys(['lnlike', 'model', 'BIC', 'best_fit_power_spectrum', 'frequencies',
↪ 'power', 'params', 'rchi2', 'probability', 'ID'])
```

we can see that numerous properties of the data and analysis have been saved for future inspection. We summarize these properties below.

- lnlike: The value of the best-fit log-likelihood for this model
- model: a string descriptor of the model that was fitted to the data
- BIC: the value of the Bayesian Information Criterion (BIC) for this model fit
- best_fit_power_spectrum: an array containing the values of the best fit model to the Fourier power spectrum
- frequencies: an array containing the frequencies for the Fourier power spectrum
- power: an array containing the Fourier power values
- params: A list of the best-fit model parameters. These parameters produce the best fit power spectrum.
- rchi2: The reduced chi-squared value, used to determine if this model is appropriate at all given the data
- probability: The probability value associated with the rchi2 value.
- ID: The ID number corresponding to the model used

2.4 Interpretation

To determine which of the fitted models is most appropriate for the data, we compare the Bayesian Information Criterion (BIC) values for each model. The BIC is closely related to the negative log likelihood, and is given by: -2 ln(L) + k ln(n), where L is the maximum likelihood, k is the number of free parameters, and n is the number of data points in the Fourier power spectrum.

The model that best minimizes BIC is considered the best fit to the data. For a model to be considered strongly preferred over other models, we typically require a difference in the BIC value of > 10 compared with other models. If two models have a very similar BIC then we do not have much evidence in favour of one over the other.

For detecting oscillations, this means we are looking for one of the models that includes an oscillation (e.g. model 1) to be strongly preferred over the models that do not contain an oscillation (e.g. model 0).

Note: the BIC values alone provide only a *relative* comparison between the selected models, not an absolute goodness of fit to the data. In other words, *all* of the models may fit poorly. To check this we include the additional parameter rchi2, which is a chi-squared like metric for exponentially distributed data. If the rchi2 value of a model is too large (as estimated via the associated probability value), it should be considered inappropriate for the data. Suggested cutoff values for the probability are $p < 0.05$ or $p < 0.01$.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index