CorrSim

A Multiwavelength Timing Simulator

How-To Guide v.1.3
29th Feb 2024

Dr. John A Paice
with Prof. Ranjeev Misra and Prof. Poshak Gandhi

1 Introduction

Hello! If you're reading this, I assume you want to try out CorrSim. Hopefully, this document will help you out with that.

There are two main ways that you can use CorrSim. One is using the Graphical User Interface (GUI), which will prompt you for all the required parameters, as well as let you load in an example; you can find instructions for that in Section 3. The other is to run it as a function from within a Python environment; you can find out how to do that in Section 4. Meanwhile, you should also find a lot of use in the Glossary (Section 5), which will let you know what each parameter does in both those methods. Detailed instructions are also included on how to define Power Spectra (Sec. 6) and Phase/Time Lags (Sec. 7). Finally, a few known technical issues are noted in Section 8.

If you have any queries (or bug reports!), please feel free to email johnapaice@gmail.com, or submit an issue/pull request on the GitLab page.

1.1 The `CorrSim` Paper

This space is reserved for when the `CorrSim` paper has been submitted and can be properly cited. In the meantime, if CorrSim has been useful in your work, please include the line:

"This work made use of the `CorrSim` software (Paice et al., in Prep)."

With a footnote linking to the appropriate GitLab page:

https://gitlab.com/astro_johnapaice/CorrSim/

2 Getting `CorrSim`

`CorrSim` is hosted on the following GitLab page:

https://gitlab.com/astro_johnapaice/CorrSim/

To get it on your system, either download or git clone it from there.

2.1 Dependencies

`CorrSim` requires the following python modules. They should all be installable using:

> `pip install [module name]`

- `copy`
- `importlib`
- `math`
- `matplotlib`
- `numpy`
- `os`
- `pandas`
- `scipy`
- `stingray`
- `time`
- `tkinter`
- `warnings`

3 How to... use `CorrSim` through the GUI

You can run the CorrSim GUI by navigating to your CorrSim folder in a terminal, and entering:

> `python3 CorrSim_GUI.py`

(Or your Python 3 version of choice. CorrSim is not compatible with previous versions of Python.)

Input parameters are set within this GUI. Parameter names can be hovered over with the mouse to bring up an explanatory tooltip, and are also described here in Section 5. You can use File -> Load/Save to load/save parameters. An example file (Example_Input_Parameters.txt) is provided.

To help you create Power Spectra and Phase/Time lags, you can test the models using the 'Test Power Spectra' and 'Test Lags' buttons. These will just plot your inputs, so you can check what you're telling the program to create.

By running the program, it will automatically create files containing:

- A log file containing your parameters
- The simulated power spectra (F_rms normalised) as a text file
- The simulated lightcurves as a text file
- The simulated lightcurves as a .png

Optionally, `CorrSim` can also give you the following:

- A plot of the red noise and the model power spectrum as a .png
- The Model CCF as both a text file and a .png
- The CCF calculated from the simulated lightcurves as both a text file and a plot.
- The Model Fourier data (Power Spectra, Coherence, and Phase/Time lags) as both a text file and a plot.
- The Fourier data calculated from the simulated lightcurves as both a text file and a plot.

`CorrSim` will also give you extra plots if multiple boxes are checked:

- A plot of the CCF calculated from the simulated lightcurves, with the model overlaid on top (If both 'Plot Model CCF' and 'Calculate CCF' boxes are checked).
- A plot of the Fourier data calculated from the simulated lightcurves, with the models overlaid on top (If both 'Plot Fourier Models' and 'Calculate Fourier' boxes are checked).
- A large compilation plot of the inputs and outputs (If all four of the above are checked).

4 How to... use `CorrSim` as a Function

You can also use the `CorrSim` function directly by calling it in python. In your CorrSim directory, import it as:

> `import CorrSim`

And then call the function:

> `CorrSim.CorrSim()`

The glossary below (Sec. 5) has a list of the parameters the function requires. All are fairly straight-forward, except for `scin_noise_A`, `scin_noise_B`, `Lorentz_params_A`, `Lorentz_params_B`, and `model_lag_array`, which are noted below.

4.1 Defining Scintillation Noise when using `CorrSim` as a function

The main `CorrSim` function does not calculate the parameter that controls the Scintillation noise, and must be given it instead. In order to calculate it, run:

> `python3 CorrSim_MakeScintillation.py`

This will prompt you for all the parameters asked for by the GUI, which you can also find in the glossary below. At the end, it will give you a number - use this as the number for the `scin_noise_A`/`scin_noise_B` parameter.

4.2 Defining Lorentzians when using `CorrSim` as a function

Lorentzian Parameters are to be given as a list. The length of this list must be divisible by 3 for Series A, and 4 for Series B. Each set of 3/4 is a Lorentzian, with the first number being the Normalisation, the second being the Width, the third being the Midpoint, and the fourth (in the case of Series B) being the Lorentzian's coherence with Series A.

Unlike in the GUI, there is no limit on the number of Lorentzians you can have when using `CorrSim` as a function. Other restrictions, such as those stated in Sec. 6.2, still apply (and the code may error if not followed).

4.3 Defining Lags when using `CorrSim` as a function

The method for creating lags is explained in full in Sec. 7. When using `CorrSim` as a function, lags are controlled by the parameter `model_lag_array`, which is created by calling the following:

> `python3 CorrSim_MakeLags.py`

This will prompt you for several parameters, all of which are either defined in the glossary or in the distribution descriptions. Parameters like the output directory and the fileprefix are included to print out your model lags, so that you can check that your inputs are correct.

As you enter distributions for your lag sections, it will inform you what is required, and will prevent you from entering invalid values. Restrictions are detailed further in Sec. 7.

When it has finished running, it will print a numpy array to the terminal; use this (or the variable it gives - `model_lag_array`) as the argument for the `model_lag_array` parameter.

When using the function, there is still a maximum of six lag sections allowed, just like in the GUI.

5 Glossary of Variables/Arguments

These are displayed in the order in which they appear in the function. The format is this:

- **Name in GUI / function_argument_name** (GUI/Function Only, if applicable) *(Type)*

  A short description of the variable/argument.

  (Default value, when calling the function.)

Not all variables/arguments will be present in both the function and the GUI. These will be noted.

- **Output Directory / output_dir** *(String)*

  Defines the output directory; e.g. 'CorrSim_Outputs' will save all outputs to the folder ./CorrSim_Outputs. This folder will be created if it doesn't already exist. *(Default: "CorrSim_Outputs")*

- **File prefix / fileprefix** *(String)*

  Defines a prefix for the filenames of all outputted files.

  (Default: "CorrSim - ")

- **Observation Length (s) / obs_length** *(Number, >0)*

  Length of simulated observation (seconds). Particularly relevant for the number of bins, which is the biggest factor in the size of the output files and the speed of CorrSim.

  (Default: 1000)

- **Time Resolution (s) / time_res** *(Number, >0)*

  Time resolution of simulated observation (seconds). Particularly relevant for the number of bins, which is the biggest factor in the size of the output files and the speed of CorrSim.

- **Optimise Bins** (GUI Only) *(Checkbox)*

  Optional; If this is checked when 'Calculate Bins' is clicked, the observation length is changed so that the number of bins is a 7-smooth number (i.e. a number with no factors greater than 7). Numbers with higher factors can greatly slow the calculations down.

  (Default: Unchecked)

- **Mean Counts/s (A) / mean_counts_per_sec_A** *(Number, >0)*

  Mean count rate in Series A (Counts/s).

  (Default: 1000)

- **Mean Counts/s (B) / mean_counts_per_sec_B** *(Number, >0)*

  Mean count rate in Series B (Counts/s).

  (Default: 5000)

- **F_rms (A) / F_rms_A** *(Number, >0)*

  Fractional RMS in Series A. Defines how variable the lightcurves are. For example: for the X-ray lightcurve of X-ray Binaries, F_rms = 0.3 in the hard state, and 0.04 in the Soft State.

  (Default: 0.3)

- **F_rms (B) / F_rms_B** *(Number, >0)*

  Fractional RMS in Series B. Defines how variable the lightcurves are. For example: For the Optical lightcurve of X-ray Binaries, F_rms = 0.1.

  (Default: 0.1)

- **Apply Red Noise (A and B) / apply_red** *(Checkbox (GUI), "A"/"B"/"AB"/"n" (function))*

  Apply Red Noise to Series A, Series B, both Series A and B, or neither.

  Red noise is also known as 'Brownian noise', and is proportional to frequency$^{(slope)}$ (See 'Red Noise Slope', below). `CorrSim` applies red noise to the power spectra before they're converted to lightcurves, using the methods of Timmer & Koenig 1995 (A&A **300**, 707):

  For each Fourier Frequency, two Gaussian distributed random numbers are drawn, with a standard deviation equal to the power spectra. These are the real and imaginary parts of the Fourier-transformed red noise lightcurve. They are then normalised, combined, and added to the Fourier-transformed parts of the original lightcurves.

  (Default: "n")

- **Red Noise F_rms / F_rms_rednoise_A** *(Number, >0)*

  Fractional RMS of the red noise for A; the larger the number, the greater the noise.

  (Default: 0.2)

- **Red Noise F_rms / F_rms_rednoise_B** *(Number, >0)*

  Fractional RMS of the red noise for B; the larger the number, the greater the noise.

  (Default: 0.03)

- **Red Noise Slope / rednoise_slope_A** *(Number)*

  Defines the dependence of the noise on frequency for A; Noise is proportional to $f^{(slope)}$, where f is the frequency.

  (Default: -2)

- **Red Noise Slope / rednoise_slope_B** *(Number)*

  Defines the dependence of the noise on frequency for B; Noise is proportional to $f^{(slope)}$, where f is the frequency.

  (Default: -2)

- **Apply Poisson Noise (A and B) / apply_poisson** *(Checkbox (GUI), "A"/"B"/"AB"/"n" (function))*

  Apply Poisson Noise to Series A, Series B, both Series A and B, or neither.

  Poisson noise, also known as 'shot noise', comes from the random nature of emitted photons. CorrSim applies Poisson noise to the lightcurves by drawing a random number from a Poisson distribution for each bin, with the mean centred on the flux in that bin.

  (Default: "n")

- **Apply Readout Noise (A and B) / apply_readout** *(Checkbox (GUI), "A"/"B"/"AB"/"n" (function))*

  Apply Readout Noise to Series A, Series B, both Series A and B, or neither.

  Readout noise comes from fluctuations in reading out charge from a CCD. This is dependent on 'read_noise_A' and 'read_noise_B', defined below.

  (Default: "n")

- **Readout Noise (e-) / read_noise_A** *(Number, >0)*

  Read Noise parameter for A, in units of electrons.

  (Default: 0)

- **Readout Noise (e-) / read_noise_B** *(Number, >0)*

  Read Noise parameter for B, in units of electrons.

  (Default: 0)

- **Apply Scintillation Noise (A and B) / apply_scintillation** *(Checkbox (GUI), "A"/"B"/"AB"/"n" (function))*

  Apply Scintillation Noise to Series A, Series B, both Series A and B, or neither.

  This noise comes from atmospheric effects, and is thus only applicable to simulations from ground-based observatories. Scintillation noise is dependant on a telescopes diameter, altitude, and exposure time, a target's altitude, the height of atmospheric turbulence, and some empirical value.

- **Telescope Diameter** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, >0)*

  Diameter of the observing telescope (metres). An increase in this value decreases the scintillation noise.

- **Telescope Altitude** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, >0)*

  Altitude of the observing telescope (metres) An increase in this value decreases the scintillation noise.

- **Exposure Time** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, < Time Resolution)*

  Exposure time of the observing telescope. This is different from time resolution, as it is does NOT include the time to read out information from the CCD. An increase in this value decreases the scintillation noise.

- **Turbulence Height** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, >0)*

  Height of turbulence in the atmosphere (metres). A typical value here is around 8000. An increase in this value increases the scintillation noise.

- **Target Altitude** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, 0-90)*

  Altitude of the source (degrees). An increase in this value decreases the scintillation noise.

- **Empirical Value** (GUI, or `CorrSim_MakeScintillation.py`) *(Number, >0)*

  An empirical coefficient that varies depending on the site of the telescope. This is defined as C_Y in Osborn et al. 2015 (MNRAS **452**, 1707), where several sites are listed. The mean value is 1.5. An increase in this value increases the scintillation noise.

  (Default: "n")

- **scin_noise_A** (Function Only) *(Number, >0)*

  Scintillation Noise parameter for B. This is calculated by running `CorrSim_MakeScintillation.py`, which will prompt you for various parameters (explained above). See also Sec. 4.1. Default values are for the New Technology Telescope (NTT) at La Silla, Chile.

  (Default: 0)

- **scin_noise_B** (Function Only) *(Number, >0)*

  Scintillation Noise parameter for B. This is calculated by running `CorrSim_MakeScintillation.py`, which will prompt you for various parameters (explained above). See also Sec. 4.1. Default values are for the New Technology Telescope (NTT) at La Silla, Chile.

  (Default: 0)

- **Plot Red Noise / plot_rednoise** *(Checkbox (GUI), 1/0 (function))*

  Plots the model power spectra, the simulated red noise, and a summation of the two for each band.

- **Plot Model CCF / plot_model_ccf** *(Checkbox (GUI), 1/0 (function))*

  Plots the model Cross-Correlation Function (A time-domain representation of how the two lightcurves correlate as a function of time lag. 1 = perfect correlation, 0 = no correlation, -1 = perfect anti-correlation).

  (Default: 0)

- **Calculate CCF / calculate_ccf** *(Checkbox (GUI), 1/0 (function))*

  Calculates and plots the Cross-Correlation Function (CCF) of the simulated lightcurves. This CCF is made by splitting up the lightcurves into segments (defined by Segment Size (s)), running CCF analysis on each of them, and then averaging the result.

  (Default: 0)

- **Segment Size (s) / ccf_segment** *(Integer)* Define the length (in seconds) the lightcurves are cut up into before the Cross-Correlation Function is calculated. Must be less than Observation Length.

  (Default: 30)

- **Binning (bins) / ccf_binning** *(Number, >0)*

  Optional; if >1, average the points of the lightcurves over this many bins before the Cross-Correlation Function is calculated. This can be done to speed up calculations.

  (Default: 0)

- **Plot Fourier Models / plot_model_fourier** *(Checkbox (GUI), 1/0 (function))*

  Plot the input models of the Power Spectra, Coherence, Phase Lags, and Time Lags, all dependant on Fourier frequency. Power Spectra are representations of how much signals vary. Coherence is how much the two signals correlate ($0 < c(f) < 1$). Phase Lags are a representation of how much each frequency is offset in phase. Time Lags are a translation of phase lags to the time domain.

  (Default: 0)

- **Calculate Fourier / calculate_fourier** *(Checkbox (GUI), 1/0 (function))*

  Calculates and plots the power spectra, coherence, and phase/time lags between the two series. It accomplishes this by splitting up the lightcurves in a number of segments (size given by 'Segment Size (bins)'), carrying out the analysis on each segment, and then averaging the result. These will be different from the inputted values, due to the effects of noise and finite sampling.

  (Default: 0)

- **Segment Size (bins) / fourier_bins** *(Number, >0)*

  Define the length (in bins) the lightcurves are cut up into before the Fourier analysis is carried out. Ideally, this should be a power of two.

  (Default: 512)

- **Rebinning / fourier_rebin** *(Number, >0)*

  Define the amount of logarithmic rebinning to do when calculating and plotting the Fourier analysis. This goes straight to the Stingray software, which uses the convention of Rebinning-1; i.e. a value of 0.1 will rebin logarithmically with a factor of 1.1.

  (Default: 0.3)

- **Reference Freq. (Hz) / reference_freq** *(Number, >0)*

  This is used when calculating the phase lags. Phase lags are inherently constrained between +/-pi. If the actual phase lags are outside of this range, e.g. between pi and 2pi, then analysis will show that they are between -pi and 0 radians due to the cyclical nature of sine waves. A 'reference frequency' is thus defined to be the frequency at which the code calibrates the rest of the phase lags; this should be the frequency at which the measured phase lag is between +/-pi.

  (Default: 1)

- **Remove White Noise / remove_whitenoise** *(Checkbox (GUI), 1/0 (function))*

  This is used for plotting the recovered power spectra and coherence. Note that this is a VERY simple approximation - it assumes that the last point in the power spectrum is white-noise dominated and removes 99% of that value from the power spectra. This is ONLY a good approximation when the spectrum is white-noise dominated at the highest Fourier frequency.

  (Default: 0)

- **'Broken Powerlaw' or 'Lorentzians' / power_model_type** *(Radio Button (GUI), "broken_powerlaw"/"lorentzians" (function)*

  Sets the type of power spectra to be used. 'Broken Powerlaw' uses a simple model of constant power that breaks at some frequency, and then becomes a power law component that decreases with increasing frequency. The coherence is described in the same way. 'Lorentzians' allows you to define a series of Lorentzians that sum to the power spectrum for each series. Series B includes an extra parameter for each Lorentzian to define how coherent it is with Series A.

  (Default: "broken_powerlaw")

- **Power Spectra Index / ps_power** *(Number)*

  Index of the Power Law component of the power spectra.

  (Default: -1.5)

- **Power Spectra Break Freq. / break_freq** *(Number, >0)*

  Break Frequency of the power law (the frequency at which it transitions from a constant to a power law).

  (Default: 1)

- **Coherence Constant / coh_constant** *(Number, >0)*

  The fraction that Series B is coherent with Series A during the constant component of the coherence.

  (Default: 0.1)

- **Coherence Index / coh_power** *(Number)*

  Index of the Power Law component of the coherence.

  (Default: -0.5)

- **Coherence Break Freq. / coh_break_freq** *(Number, >0)*

  Break Frequency of the coherence (the frequency at which it transitions from a constant to a power law)

  (Default: 0.5)

- **Lorentz_params_A** (Function Only) *(List)*

  Lorentzian Parameters for Series A. This must be a list divisible by 3. Each set of three is a Lorentzian, with the first number being the Normalisation, the second being the Width, and the third being the Midpoint. See also Sec. 4.2.

  (Default: [])

- **Lorentz_params_B** (Function Only) *(List)*

  Lorentzian Parameters for Series B. This must be a list divisible by 4. Each set of four is a Lorentzian, with the first number being the Normalisation, the second being the Width, the third being the Midpoint, and the fourth being the Lorentzian's coherence (between 0 and 1) with Series A. See also Sec. 4.2.

  (Default: [])

- **Lorentzian Parameters (Norm, Width, Mid, Coh)** (GUI Only) *(Numbers, >0)*

  This array of boxes produces the Lorentzians. Up to six can be used in each series. Each Lorentzian is defined by:

$$L(f) = \frac{N}{\pi} \frac{\frac{W}{2}}{(f-M)^2 + (\frac{W}{2})^2} \tag{1}$$

  Where f is frequency, and N, W, and M are the Normalisation (Norm), Width, and Midpoint (Mid) respectively. The Series B Lorentzians have an extra parameters, Coh, which defines how coherent each Lorentzian is with Series A. These are explained in more detail in Sec. 6.2.

- **'Time' or 'Phase' / time_or_phase** *(Radio Button (GUI), "time"/"phase" (function)*

  Lags can be defined either as time lags or phase lags. Time lags are defined in log-log space, while phase lags are defined in semi-log space.

  (Default: "phase")

- **Overall Lag / overall_lag** *(Number, >0)*

  The time/phase lag to use for all frequencies not otherwise defined.

  (Default: 0)

- **model_lag_array** (Function Only) *(ndarray)*

  Array for the lag parameters. This requires running `CorrSim_MakeLags.py`, which will prompt you for several values. When it has finished running, it will print an numpy array to the terminal - use this (or the variable it gives - `model_lag_array`) as this argument. See also Sec. 4.3.

  (Default: np.zeros((6, 7)))

- **Time/Phase Lag Parameters** (GUI Only) *(Number)*

  Lags are defined by defining several sections of different distributions. Think of it as 'drawing' the lag plot; define the distribution you want (e.g. Linear, Polynomial), the starting frequency, the ending frequency, and then any required lags.

  See Section 7 for more information.

  Note that the program needs 0 values for all unused boxes, and will often fail if not provided them.

- **write_data** (Function Only) *(1/0)*

  Writes the data to text files. For the GUI, this is always enabled.

  (Default: 1)

6 How to... define Power Spectra

A Power Spectrum is a representation of a lightcurve in Fourier space - or, in simpler terms, it's a way to describe (or show) the variability of a lightcurve. This is done by splitting the lightcurve up into a series of sine waves of different frequencies, and then finding the amplitude of each of those sine waves - a higher amplitude (or 'Power') means more variability at that frequency.

In CorrSim, Power Spectra can be defined in two main ways. A simple method is to use broken powerlaws; these are described in Section 6.1. A more in-depth method is to use Lorentzians; these are described in Section 6.2.

6.1 Using Broken Powerlaws

Broken Powerlaws feature a flat top (i.e. constant variability with increasing frequency) until a 'break frequency', at which point it transitions to a power law, typically with a negative index (i.e. decreasing variability with increasing frequency). This is a simple approximation to the power spectrum of many astrophysical sources.

In `CorrSim`, a single broken powerlaw is defined for both Series A and Series B. Example power spectra defined by broken powerlaws can be seen in Figure 1, with the parameters that define them being shown on the left of Figure 2.

The difference between the two lightcurves is thus only defined by the coherence. The fraction that Series B is coherent with Series A is also defined using a broken powerlaw; since this is multiplied by the broken powerlaw, it has a more complex shape with both break frequencies affecting it. Figure 1 shows a demonstration of this.

6.1.1 Requirements and Restrictions:

- The Power Spectral Break Frequency must be >0.
- The Coherence Constant must be ≥ 0.
- The Coherence Break Frequency must be >0.

6.2 Using Lorentzians

Lorentzians are a type of distribution which, when summed together, can create structures very similar to observed power spectra. They are described by the following equation:

$$L(f) = \frac{N}{\pi} \frac{\frac{W}{2}}{(f - M)^2 + (\frac{W}{2})^2} \tag{2}$$

where f is frequency, and N, W, and M are the Normalisation (Norm), Width, and Midpoint (Mid) respectively.

Lorentzians come in two main flavours - zero-centred Lorentzians have a midpoint equal to zero, and in power spectra, can describe the overall shape of the distribution. Lorentzians that do not have a midpoint equal to zero can often be used to describe periodic features that occur at certain frequencies - physically, this could be the rotation period of a pulsar, or perturbations in a rotating accretion disc.

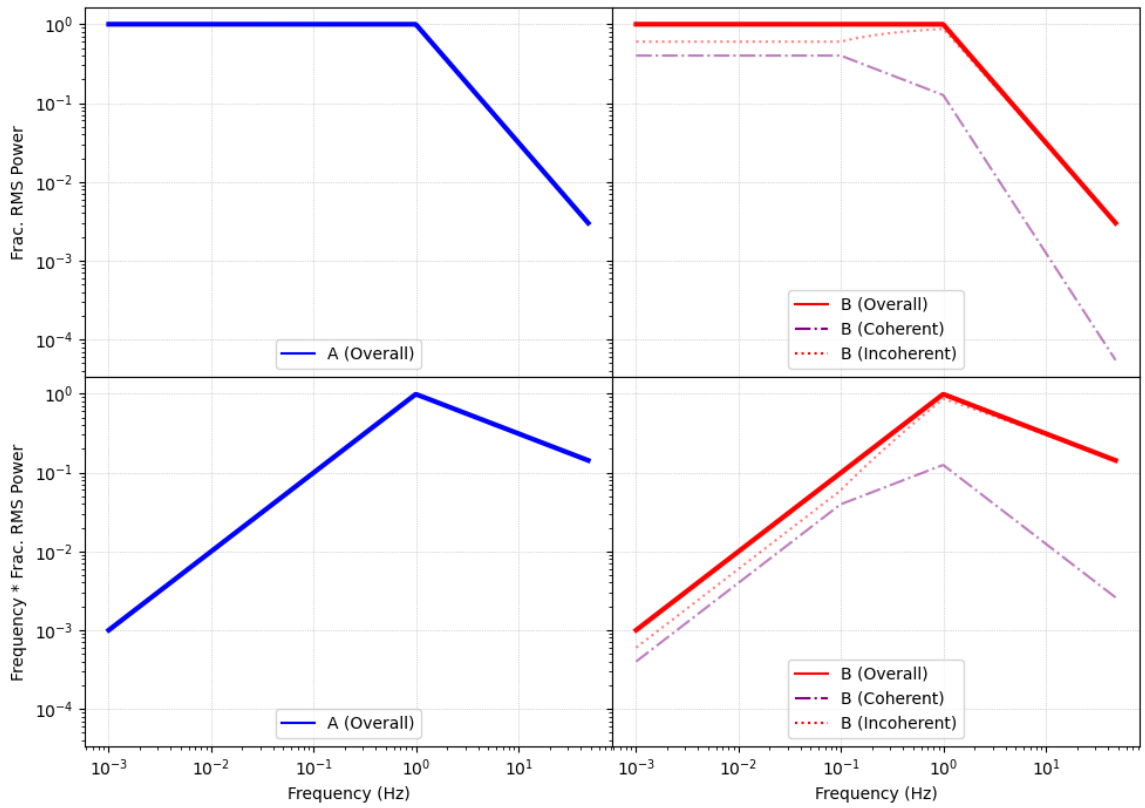

Figure 1: Example power spectra (thick lines), each made up of a broken powerlaw, defined by the values seen in Fig. 1. The top plots show Fractional. RMS power, while the bottom plots show Frequency × Fractional RMS power. The left plots show the Series A power spectrum, while the right plots show the Series B power spectra, including the coherent power spectrum as a purple dashed-dotted line, and the incoherent power spectrum as a red dotted line. Note how the coherence, which follows its own broken powerlaw, is also affected by the break in the overall power spectrum.

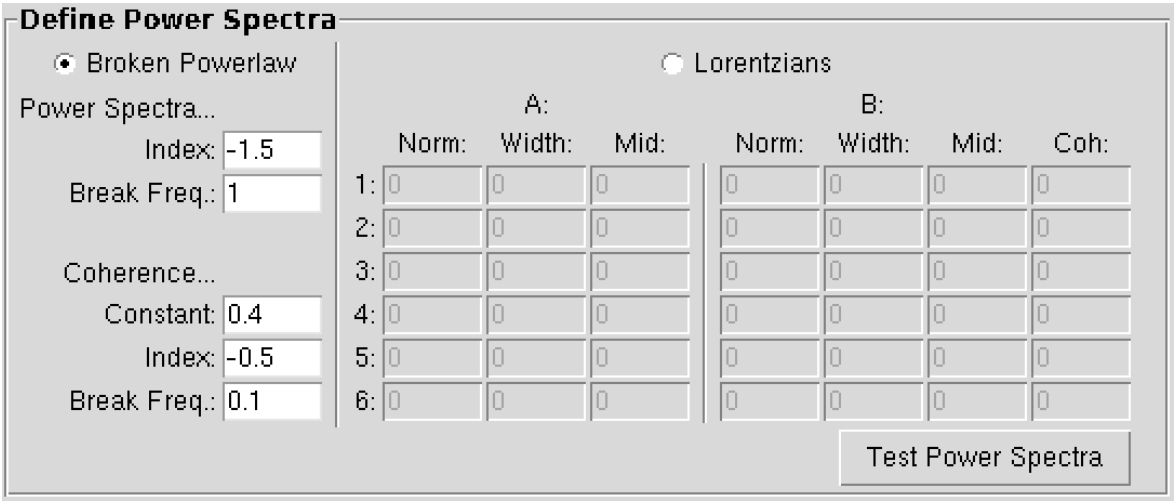

Figure 2: GUI screenshot of the parameters for the Broken Powerlaw plots in Fig. 1.

`CorrSim` allows up to six Lorentzians to be defined for each series, and unlike with Broken Powerlaw, each Series is fully independent. They are related to each other only be an additional parameter for Series B Lorentzians, 'Coh', which defines how coherent that Lorentzian is with Series A. The coherence is thus the summation of all these coherent power spectra.

Example power spectra defined by broken powerlaws can be seen in Figure 3, with the parameters that define them being shown on the right of Figure 4.

6.2.1 Requirements and Restrictions:

- If 'Lorentzians' is selected for the power spectral type, then at least one Lorentzian in both Series A and B must be specified.
- All Normalisations must be ≥ 0.
- All Widths must be ≥ 0.
- All Coherences must be $1 \geq \text{Coh} \geq 0$.
- All Lorentzians with Normalisation >0 must also have Width >0, and vice versa.

7 How to... define Lags

Lags define how much the two lightcurves relate to each other in time. They are both described in the Fourier domain; i.e. if you were to split each lightcurve up into its component sine waves, the lags would tell you how much the sine waves in Series B lag (or precede) Series A. These can be defined as Phase Lags (i.e. how much each Fourier frequency lags as a function of the phase of that frequency; Sec. 7.1), or as Time Lags (i.e. how much each Fourier frequency lags as a function of time; Sec. 7.2). Whichever regime is defined, `CorrSim` automatically calculates and gives the other regime using the following equation:

$$\delta_\tau = \frac{\delta_\phi}{2\pi f} \tag{3}$$

where δ_τ is the time lag, δ_ϕ is the phase lag, and f is the frequency of the bin.

You define these lags by giving `CorrSim` a series of distributions and co-ordinates. You can think of this like 'drawing' the phase lags onto the plot. Five distributions are available, which are all described by up to six parameters (Freq_1, Lag_1, Freq_2, Lag_2, Freq_3, Lag_3).

- **Const. Time**

  Sets a constant lag in time (Lag_1, in seconds) between Freq_1 and Freq_2.

  (Requires: Freq_1, Lag_1, Freq_2)

- **Const. Phase**

  Sets a constant lag in phase (Lag_1, in radians) between Freq_1 and Freq_2.

  (Requires: Freq_1, Lag_1, Freq_2)

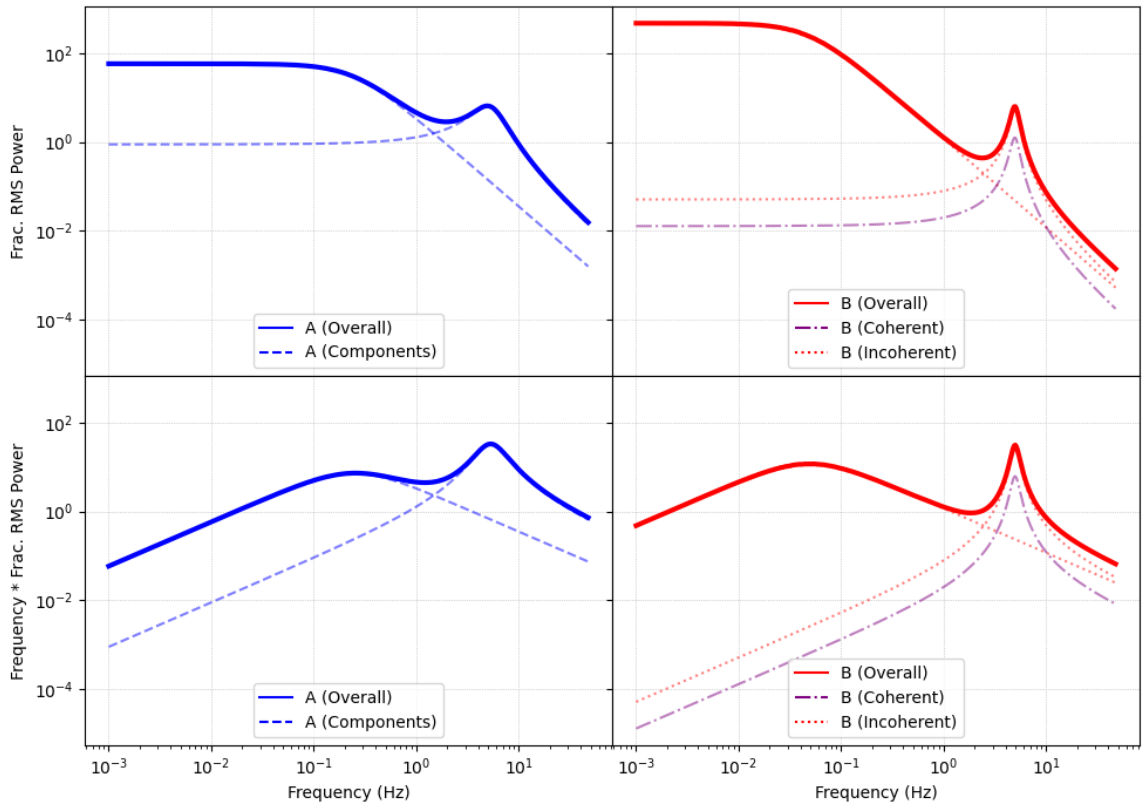

Figure 3: Example power spectra made of up Lorentzians (thick lines) for Series A (left) and Series B (right), shown with both Fractional RMS normalisation (Top) and Frequency × Fractional RMS normalisation (Bottom). Component Lorentzians for Series A are shown as dashed lines. Coherent components for Series B are shown as purple dash-dotted lines, while incoherent components are shown as red dotted lines.

Define Power Spectra

Broken Powerlaw — Lorentzians

Power Spectra...
Index: -1.5
Break Freq.: 1

Coherence...
Constant: 0.1
Index: -0.5
Break Freq.: 0.1

	A: Norm:	A: Width:	A: Mid:	B: Norm:	B: Width:	B: Mid:	B: Coh:
1:	45	0.5	0	75	0.1	0	0
2:	40	4	5	10	1	5	0.2
3:	0	0	0	0	0	0	0
4:	0	0	0	0	0	0	0
5:	0	0	0	0	0	0	0
6:	0	0	0	0	0	0	0

Test Power Spectra

Figure 4: GUI screenshot of the parameters for the Lorentzians shown in Fig. 3. Note that the four unused Lorentzians still require 0 values to be entered.

- **Linear**

  Sets a linear lag between coordinates (Freq_1, Lag_1) and (Freq_2, Lag_2).

  (Requires: Freq_1, Lag_1, Freq_2, Lag_2)

- **Power**

  Sets an exponentially increasing (or decreasing) lag between coordinates (Freq_1, Lag_1) and (Freq_2, Lag_2).

  (Requires: Freq_1, Lag_1, Freq_2, Lag_2)

- **Polynomial** Solves and plots a second-order polynomial for the three coordinates given: (Freq_1, Lag_1), (Freq_2, Lag_2), (Freq_3, Lag_3).

  (Requires: Freq_1, Lag_1, Freq_2, Lag_2, Freq_3, Lag_3)

These distributions are demonstrated in Fig. 5 for phase lags, and Fig. 7 for time lags.

Note that continuity between distributions is not enforced - if you wish there to be no discontinuities in the lags, this must be implemented manually.

7.1 Using Phase Lags

Phase lags are defined in units of radians. There are 2π radians in one complete oscillation of a frequency, and are also symmetrical - 0 and 2π are both mathematically identical - so phase lag plots here are defined between $-\pi$ and $+\pi$. For this reason, the model phase lag plots shown in Fig. 5 & 7 also include the phase lags offset by -4, -2, +2, and $+4\pi$.

Note that phase lags are plotted in semi-log space (with the log being in frequency). Hence, all distributions in phase lags are similarly in semi-log space (i.e. a linear distribution in phase lags is not linear in normal space).

Figure 5 shows example phase lags that demonstrate all distributions, while Figure 6 shows the parameters that were used to create them. These parameters are also included as an example file with the `CorrSim` download.

7.1.1 Requirements and Restrictions:

- All distributions must be defined in order of increasing frequency (Freq_1).
- If any distributions overlap, earlier distributions will take precedence.
- For all distributions, the required frequencies must be >0. This does not apply to the unused frequencies (i.e. Freq_2 in the constant distributions, or Freq_3 in all distributions but polynomial).
- In all non-constant distributions, Freq_2 > Freq_1.

7.2 Using Time Lags

Time lags are defined in units of seconds. Time lags can be be both positive (Series B lags Series A) or negative (Series A lags Series B). Both positive and negative time lags are plotted in Fourier products.

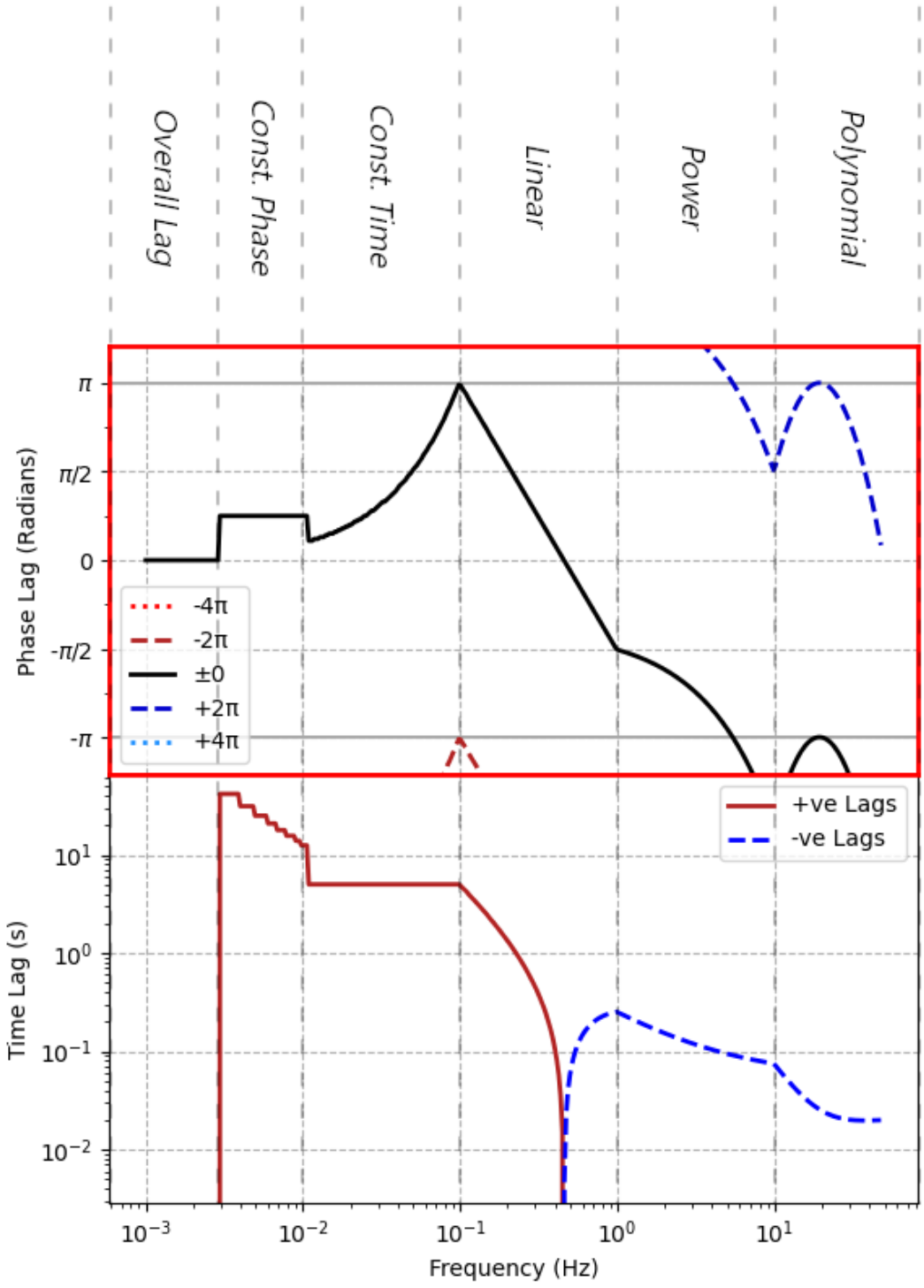

Figure 5: Lags defined by Phase, showing all five distributions as well as an overall lag. The red box indicates that these have been defined by phase. Note how the phase lags 'wrap around' beyond $\pm\pi$, and how some distributions change their shape when converted to time lags. The parameters for the lags are shown in Fig. 6.

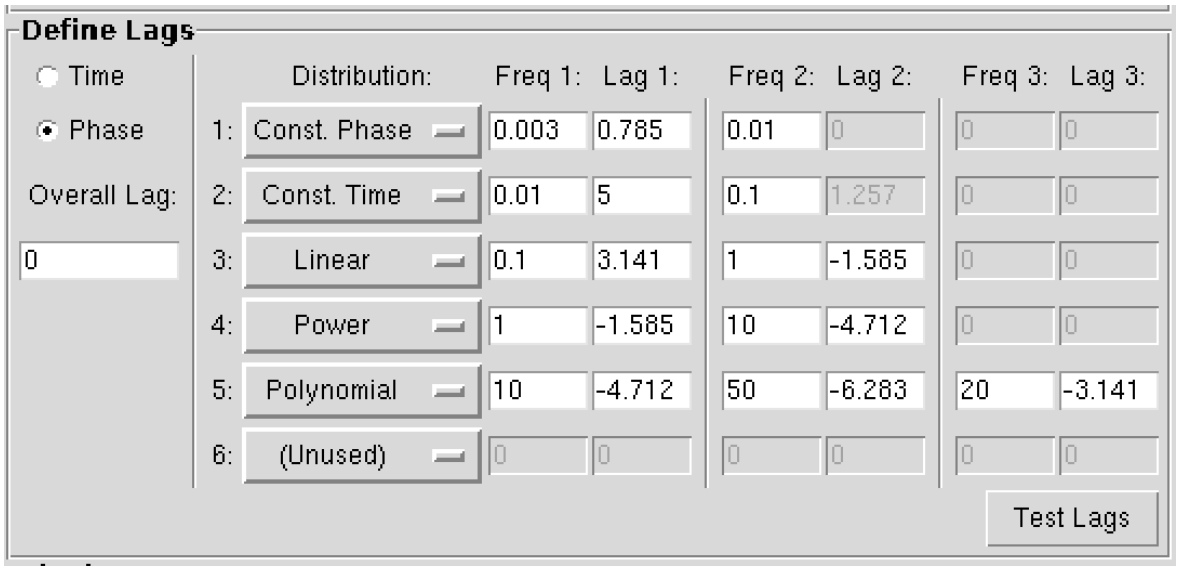

Figure 6: GUI screenshot for the phase lag parameters shown in Fig. 5.

Note that time lags are plotted in log-log space. Hence, all distributions in time lags are similarly in log-log space.

Figure 7 shows example time lags that demonstrate all distributions, while Figure 8 shows the parameters that were used to create them. These parameters are also included as an example file with the `CorrSim` download.

7.2.1 Requirements and Restrictions:

- All distributions must be defined in order of increasing frequency (Freq_1).
- If any distributions overlap, earlier distributions will take precedence.
- For all distributions, the required frequencies must be >0. This does not apply to the unused frequencies (i.e. Freq_2 in the constant distributions, or Freq_3 in all distributions but polynomial).
- In all non-constant distributions, Freq_2 > Freq_1.
- Unique to time lags, distributions can neither straddle nor go to zero - e.g. you cannot have a linear distribution between +3 s and -3 s, or between +3 s and 0 s. However, two separate distributions can be defined on either side of zero, each going to a small, non-zero number (e.g. one distribution between +3 s to +0.01 s, and the next between -0.01 s and -3 s). This restriction does not apply to unused Lag parameters.

8 Known Issues

Nothing is perfect, and oh boy does that include CorrSim! Here are a few technical issues that have been noticed. Please contact me if you have any issues, or even if you know of how to solve any listed here.

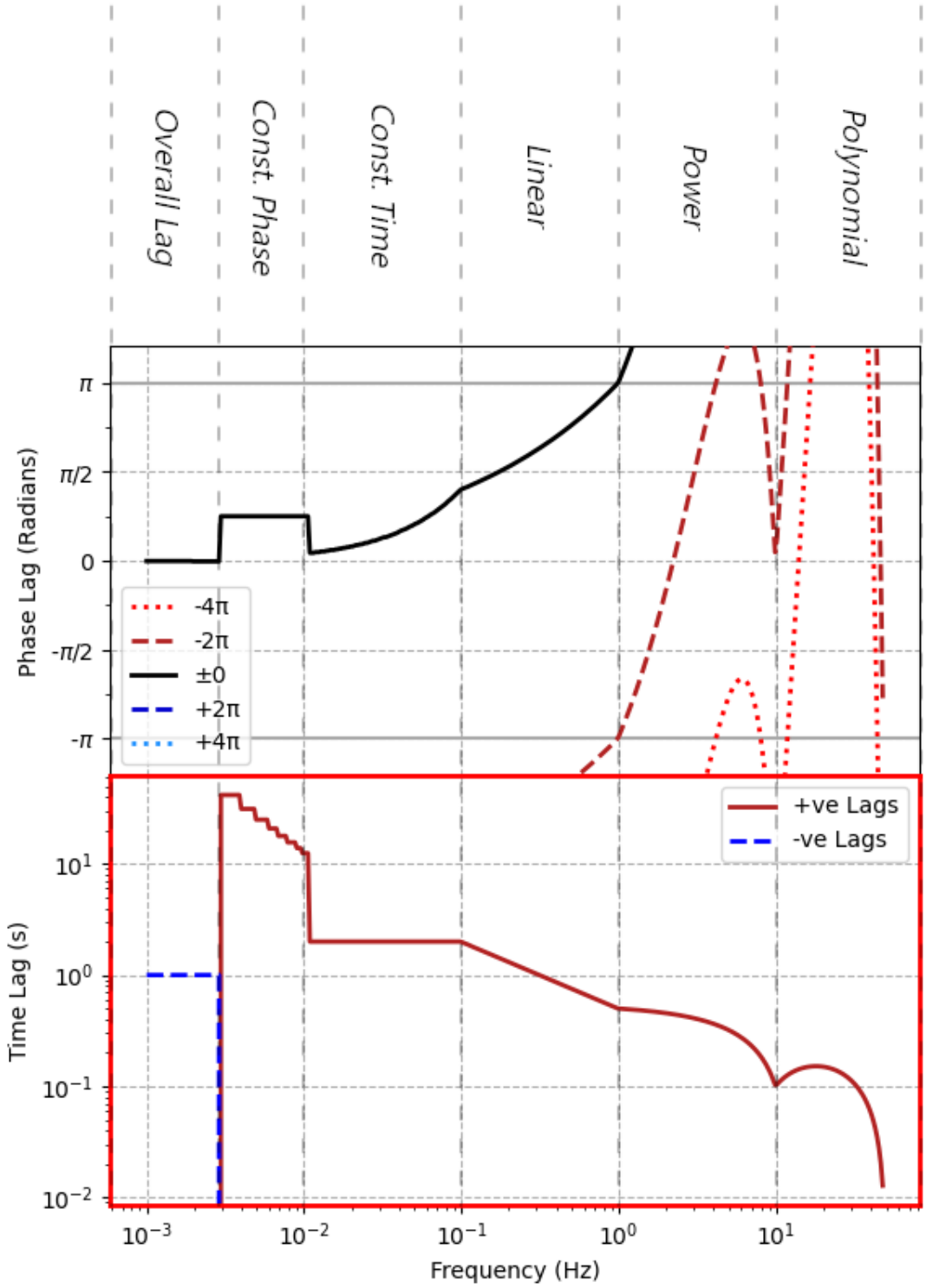

Figure 7: Lags defined by Time, showing all five distributions as well as an overall lag. The red box indicates that these have been defined by time. Note how higher-frequency time lags can show runaway phase lags. The parameters for the lags are shown in Fig. 8.

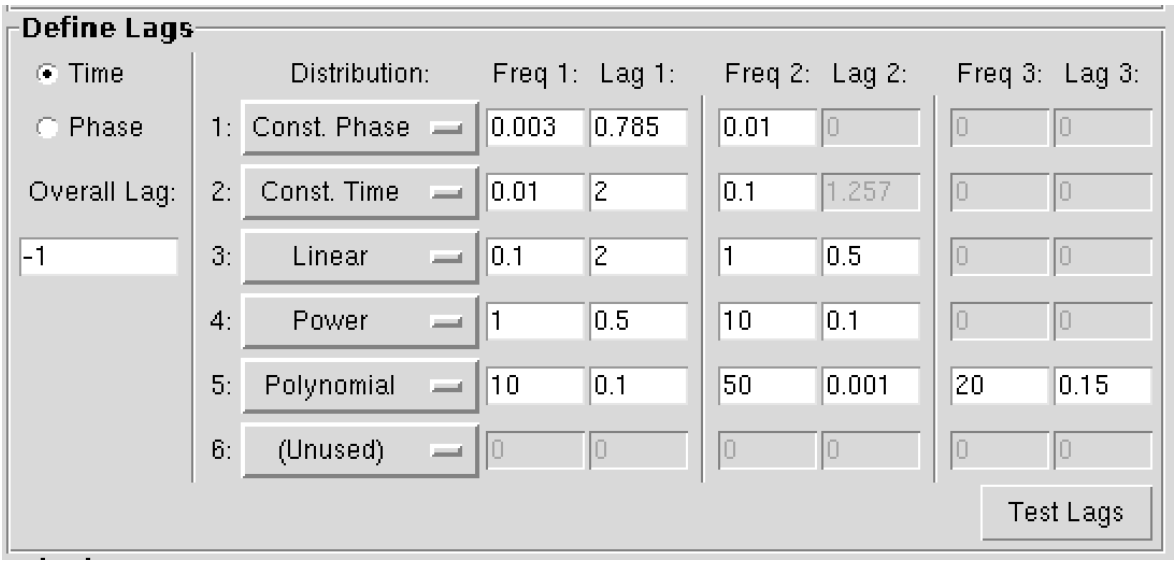

Figure 8: GUI screenshot for the time lag parameters shown in Fig. 7.

- **The GUI window is the wrong size.** This is something that's system/OS dependant, and I don't know how to dynamically resize the window! You should be able to resize it yourself; else, open `CorrSim_GUI.py` and go to the line `self.geometry("1115x850")`. Change those numbers to whatever pixel resolution works for you!
- **Some GUI buttons occasionally do not work.** This may be a problem with `tkinter`, the module used to render the GUI. Try moving the window or clicking the button few more times. Hitting 'Calculate Bins' has also been known to solve this issue.
- **The 'File' option in the Menu bar does not appear if the GUI is opened from `ipython` a second time.** This may also be a problem with `tkinter`, but the reason is unknown. To avoid this, either restart ipython, or just launch it from python directly (using, e.g. '`python3.11 CorrSim_GUI.py`')
- **The time/phase lags are inverted (i.e. a factor of -1 out).** It's unknown where this came from, but it was noticed after changing systems and python installs. It could be that some versions of the underlying FFT software calculate time lags differently, as that was a new warning that came in when systems were changed. If your time lags/phase lags are inverted from what you think they should be, open up `CorrSim_Functions.py` and go to the line `time_lags, time_lags_err = cross_spectrum_binned.time_lag()` - change the line after it to invert (or un-invert) the time lags.

9 Changelog

- *v1.0: 27th May 2022* - Original Version
- *v1.1: 18th November 2022* - Updated to include new parameters for the Red Noise.
- *v1.2: 13th December 2022* - Updated author order, dependencies (now includes copy), and parameters (included a new checkbox for plotting the input Red Noise).

  - *v1.2.1: 14th December 2022* - Updated Section 3 with information on the red noise plot. Also clarified the 'Optimise Bins' parameter and updated the 'Apply Red Noise' parameter. Additional typographical fixes also made.

- *v1.3: 29th February 2024* - Added a 'Known Issues' section for possible currently-unresolved problems with the program.