Generation of Noisy Atomic Clock Data for GPS.DM Dark Matter Observatory Simulator

by

Alex Rollings

Thesis Advisor: Dr. Andrei Derevianko

A senior thesis submitted in fulfillment of the requirements for the Bachelor of Science degree in Physics

University of Nevada, Reno

May, 2016

Abstract

Evidence shows that approximately 27% of the mass-energy budget of the universe is composed of dark matter[1]. Despite its prevalence, and the evidence supporting its existence, it has never been directly observed. However, the theoretical groundwork for dark matter in the form of a topological defect in space is believed to cause a "transient-in-time change in fundamental constants[2]". Variation of fundamental constants would affect the atomic transition frequency of an atomic clock. Such a change might be detectable by cross comparing the phase data between atomic clocks on board GPS satellites. This data however, contains fluctuations in the form of noise. Thus, to confidently identify a dark matter signal, one must be sure that it can be detected among the inherent noise. The objective of this research is to characterize and reproduce the types of noise observed in atomic clocks. The two main types of quantum oscillators under consideration are that of Cesium and Rubidium. The types of noise prevalent in these oscillators are: White PM, Flicker PM, White FM, Flicker FM, and Random Walk FM (PM and FM stand for phase and frequency modulations). Each type of noise follows a frequency power law, as given by the Allan variance, and can be characterized by their respective power spectral densities. These spectra are used to generate random phase data that fits each type of noise with *Mathematica* software. This data can be used to build a simulation of dark matter events that can be referenced when considering candidate signals in real clock data.

Acknowledgments

I would like to express my gratitude towards Dr. Andrei Derevianko for welcoming me onto his research team. He has devoted a great deal of time to ensure that I was successful in completing this project. I would also like to acknowledge Dr. Ben Roberts and Dr. Geoff Blewitt for meeting with me personally to help me better understand key points of the research. I would finally like to thank the Nevada Space Grant Consortium and the National Science Foundation. The funding I have received from the Nevada Space Grant has allowed me to dedicate the time and resources I need to focusing on this research, and has also given me the opportunity to travel to present my work at conferences.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Atomic clocks are instruments used to count a specific number of oscillations and convert that value into time. The problem with this principle is that counting is really all the clocks do. Time is calculated by multiplying some counted number of oscillations with the known period of oscillation. And so if the period is off, or fluctuating, so too will the time the clocks give be inaccurate. For this reason, data from atomic clocks on board Global Positioning System (GPS) satellites are compared with ground station clocks on Earth. Any discrepancy between the two is attributed to noise in the clock on board the satellite. These differences are what is referred to as the clock's *bias*, meaning error from actual value. There is the possibility that there can be more than noise affecting the clock bias. In a paper published by Derevianko and Pospelov, a theory is proposed in which dark matter in the form of a topological defects will cause a "transient-in-time change in fundamental constants[2]". Part of this effect is a change in the electron emission period of atoms within atomic clocks. A change in this frequency would be detectable in the form of a distinct clock bias. As a wall of dark matter passes through a satellite, the clock on board becomes desynchronized with the reference clock on Earth. Some period of time later, the wall of dark matter will pass through the ground station clock, and the times will once again be synchronized. Figure 1.1 illustrates the motion of the wall.

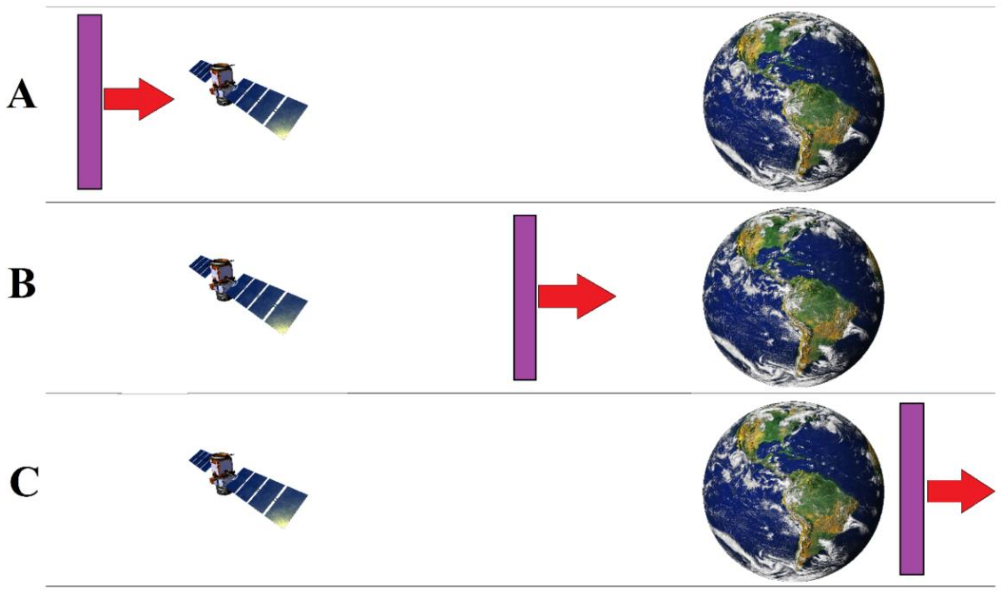


Figure 1.1*: A wall of dark matter passing through Earth. Here, the simplest case of a thin walled event is assumed. It can be seen that a sufficiently large event will first pass through satellites before reaching the Earth.*

The duration of time that the bias exists between the satellite clock and the reference clock is equal to the time it takes the event to travel the distance separating the two clocks. The average velocity of Earth through the dark matter halo is taken to be 300 km/s. At this velocity, there should be a significant jump in clock bias that lasts for up to 150 seconds[2]. This signature will take the form of Figure 1.2.

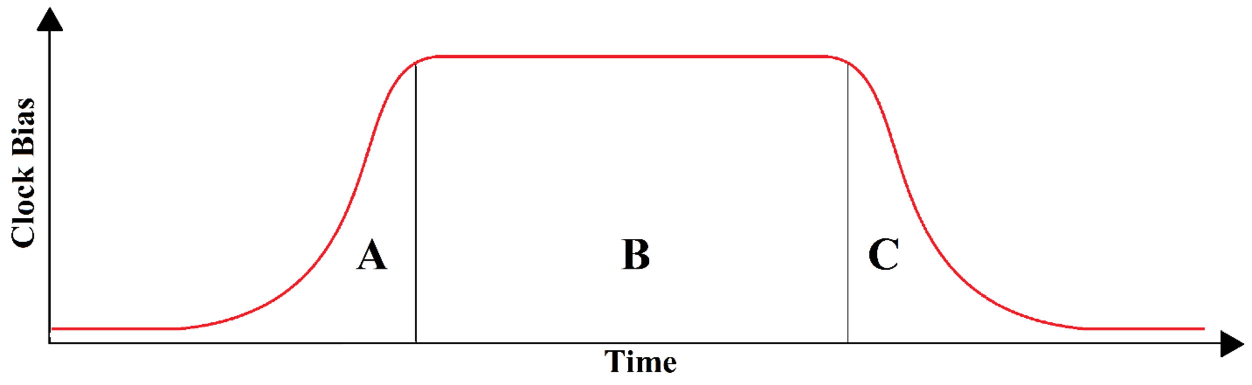


Figure 1.2: *The signature bias between a single ground station clock and a GPS satellite affected by topological dark matter.*

It seems reasonable that such a distinct event should be easy to detect. But this is assuming that the bias at all other times outside an event are equal to zero. Unfortunately, this will never be the case. The data is often riddled with too many fluctuations in the form of noise. An example of real clock bias is graphed in Figure 1.3.

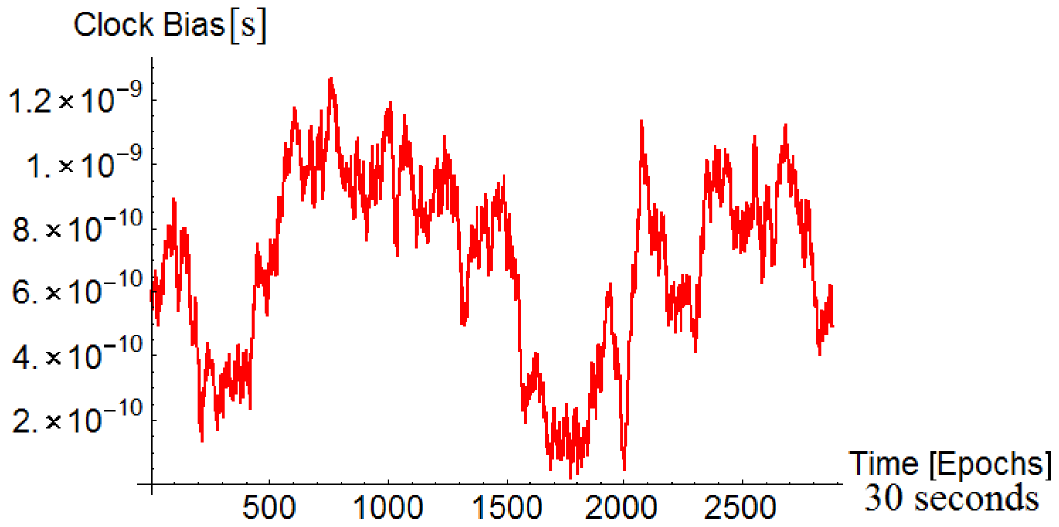


Figure 1.3: *Atomic Clock Bias from December 30, 2001. This type of data is made readily available by the Jet Propulsion Laboratories (JPL).*

The satellite data in Figure 1.3 comes from Space Vehicle Number (SVN) 22. The type of clock on this satellite for this date is Cesium. The data is sampled every 30 seconds. This corresponds to the duration of time between each GPS epoch. This means that the proposed dark matter signature (lasting for 150 seconds), would only be present for 5 of the 2880 data points plotted above. Furthermore, this image also reveals the prevalence of noise in the clock bias. This leaves the following question: Can the dark matter signature be distinguished from the noise? That question is the motivation for this thesis. In order to confidently extract such a signature, one must place bounds on the types of noise seen in Figure 1.3. This can be accomplished through spectral analysis of generic noise types. If successfully applied to the generic noise types, the methods developed can then be used to extract a dark matter signature from generated clock data. Thus a library of event models can be built, and any dark matter candidates discovered in real clock data can be compared to the existing simulation.

1.2 Dark Matter

Observations made by the Hubble Space Telescope have shown that the universe is expanding at an accelerated rate. The cause of this is generally attributed to *dark energy* whose source is unknown. This energy is believed to comprise roughly 68% of the universe's total mass-energy budget. Another 5% goes to all of the ordinary matter astronomers can account for, such as planets, stars, and nebulae. Astronomical observations indicate that the remaining 27% of the known universe is made up of matter which has not yet been detected[1].The most well-known observational evidence for dark matter comes from galactic rotation curves. Kepler's third law formulates the relationship between the orbital speed of a mass relative to its radial distance from the object it encircles. Gravity is the force that acts as a tether between the two objects, and its strength is diminished with distance. As a result, the circular speed of the orbiting mass decreases with distance as well. This principal can be applied to the motion of stars orbiting about the center of galaxies. Because galaxies are denser toward their centers, it is expected that the orbital speed of masses within the galaxy should be inversely proportional to their distance from the galactic center. However, it is often found that this is not the case. In fact, orbital speed has even been seen to increase with radial distance. The discrepancy is believed to be caused by the gravitational pull of the aforementioned 27% of undetected matter. Because it does not emit radiation in any range of the electromagnetic spectrum, it has been aptly named *dark matter*[3].

A recent theory has been proposed in which dark matter is not a particle (which has been the common view), but rather a collection of topological defects in space. The fundamental constants belonging to regular matter passing through this defect will temporarily shift as a result of interaction. The frequency of light emitted through electron transitions is sensitive to this type of change. This is why atomic clocks on board GPS satellites would form a 50,000 km aperture for detecting topological dark matter[2]. There are, on average, 24 GPS satellites in operation at any given time[4].

1.3 Atomic Clocks

Atomic clocks are like other time keeping devices, in that they count some number of oscillations to tell time. They are special; however, because they can tell time with greater precision and accuracy than any type of instrument ever built. An often quoted statement is that it would take over a million years for the time held by a modern Cesium clock to drift by one second[5]. To understand the fundamental principles of clocks, atomic or not, one must first understand how to measure time.

1.3.1 Basic Principles of a Clock

Time is a quantity defined by people. It is relative. The universe does not know what time it is, people do. The notion of the second has changed over time, but has always been measured by counting periodic cycles. A physical system that undergoes periodic cycles is often called an *oscillator*. A pendulum is an example of an oscillator. It begins in one state and retraces a given path only to return to that initial state. Consider a basic periodic function such as the one shown in Figure 1.4. It is a sine wave, and takes the following form:

$$Y(t) = Y_0 \cdot \sin[2\pi\nu_0 t]\,. \qquad (1.1)$$

Here, Y_0 is the amplitude of the signal. It corresponds to the maximum height of the sine wave. The duration of time that the wave is analyzed is measured in t seconds. The distance that the wave travels before repeating is denoted by the wavelength λ. Finally, the wave propagates with a frequency of ν_0, which has units 1/s. The entire argument of the sine function gives the *phase*, which is how far into a given cycle the signal is. In the corresponding figure below, Y_0 and ν_0 have both been assigned a value of 1.

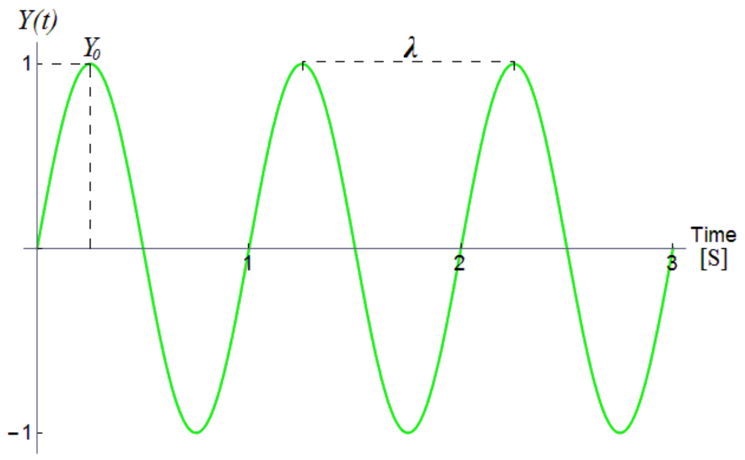


Figure 1.4: *A sine wave.*

The physical interpretation of frequency defines how many full cycles, or revolutions, a periodic function can complete in one second. Conversely, the time it takes the function to complete one full cycle is simply the inverse of the frequency. This is known as the *period* of oscillation, *T*, and is written as

$$T = \frac{1}{\nu_0}. \tag{1.2}$$

The periodic functions in question take the form of sines and cosines. Since both of these functions complete one full cycle in 2π radians, it is useful then to express their frequencies in terms of radians. This is known as the *angular frequency*, and it is defined as

$$\omega_0 = 2\pi\nu_0 = \frac{2\pi}{T}, \tag{1.3}$$

which has units of radians/second. Counting the time elapsed then becomes trivial. All one has to do is count the number of completed cycles, and either multiply this number by the oscillation period, or divide by the known frequency:

$$Time\ elapsed = number\ of\ cycles \cdot T = \frac{number\ of\ cycles}{\nu_0}. \tag{1.4}$$

1.3.2 The Quantum Oscillator

Electrons that are bound to atoms can only exist in discrete energy levels. Each level has the potential to split into further divisions in the presence of a magnetic field. This is known as the Zeeman effect, and it results in the hyperfine structure of the atom due interaction between the electron spin and the nuclear spin. When an electron transitions from a higher energy state to a lower one, it emits a photon. The frequency of the electromagnetic wave emitted corresponds directly to the energy difference between the two levels. Likewise, an electron can absorb a photon of the same frequency, allowing it to transition to a higher energy state[6]. This principle forms the foundation of how an atomic clock works.

Each clock contains atoms with a well-defined transition frequency between a given energy level. The most common type is Cesium. It has a $4.14 \times 10^{-5}\ eV$ energy gap between two hyperfine ground states. The frequency of light associated with this transition is 9,192,631,770 Hz[7]. The remarkable thing is that this number is the same for all Cesium atoms. It is so precise, that the SI unit of the second is defined as 9,192,631,770 periods of radiation corresponding to this transition[8]. There are methods by which a clock can operate. In a particular example, atoms within an atomic clock are driven by a microwave source whose frequency is fine-tuned by a feedback mechanism within the clock. The frequency of the microwaves is matched to the transition frequency described above. The atoms travel down a chamber where they are deflected by a magnetic field. Within the magnetic field, atoms in a higher energy state will deviate from their path more so than atoms in the ground state. A detector is placed in way of the excited atom's path. This detector is what provides the feedback to the microwave source. The goal is to maximize the number of excited atoms reaching the detector by changing the frequency of the microwave source accordingly. The phase of the source is sampled and compared with ground station clocks on Earth.

A brief fluctuation in the optimal transition frequency, and thus, the phase of the oscillator, will result in a clock bias that resembles the one portrayed in Figure 1.2.

1.4 Types of Signals

The output of an atomic clock is transmitted back to Earth via a carrier signal. Thus, it becomes important to delineate between two main types of signals. They are classified as being either *discrete* or *continuous*. A continuous signal is one that is defined over all points in time. When plotted as a function of time, there are no breaks in the data. A discrete signal is the result of sampling a continuous process at different times. If the sampling interval is constant, then the plotted data will be evenly spaced by the time period between measurements. In this paper, the interval of time between samples is denoted by τ_0. The importance of each classification will become more apparent, as later mathematical development will depend on which case is examined. See Figure 2.2 for an example of each type of signal.

Whether discrete or continuous, signals can also be described as *stochastic* or *deterministic*. A deterministic signal is one whose future value can be determined based off of current trends. An example of this is the sine wave described by Equation (1.1). The value of this wave can be readily calculated at any point in time. In opposition to this are signals that are stochastic. These types encompass data sets composed of random values. An example of a stochastic process would be the noise produced in an atomic clock's signal. It would not be possible to guess the exact value of the noise at a given future time. The best that can be done in this situation is to estimate the value by examining the statistical properties of the noise.

1.5 Statistical Development

In order to ascertain the properties of the noise introduced into a signal, it will be necessary to develop some of the statistics used to characterize a given set of data. The terms listed below can be used to describe both continuous and discrete systems. However, the data analyzed in this research is discrete, and so only the discrete mathematical formulations have been given.

1.5.1 Expectation Value

The *expectation value* of a given function is the weighted probabilistic average of a function. This term does not necessarily imply that it is the most probable value. In general, the expected value of a discrete function $f(x)$ is given by:

$$E\{f(x)\} = \sum_x f(x) \cdot P(x), \tag{1.5}$$

where $P(x)$ is the probability distribution. In the limit as $x \to \infty$, the expectation value approaches the *mean* of the data[9]. Specifically, the *arithmetic mean* refers to a number that can replace all values in a list without changing the sum of the list. For a list that is N terms long and has the values $X = \{x_1, x_2 \ldots x_n\}$, the mean can be found the following way:

$$\bar{X} = \frac{1}{N}\sum_{n}^{N} x_n \,. \tag{1.6}$$

1.5.2 Variance

Though useful in their own regards, the mean, median, and most probable value are not sufficient for complete statistical interpretation of a given set of data. For example, two different sets of data might share the same mean, median, and most probable value; however, the data might be distributed differently across a range of values, as is the case in Figure 1.5.

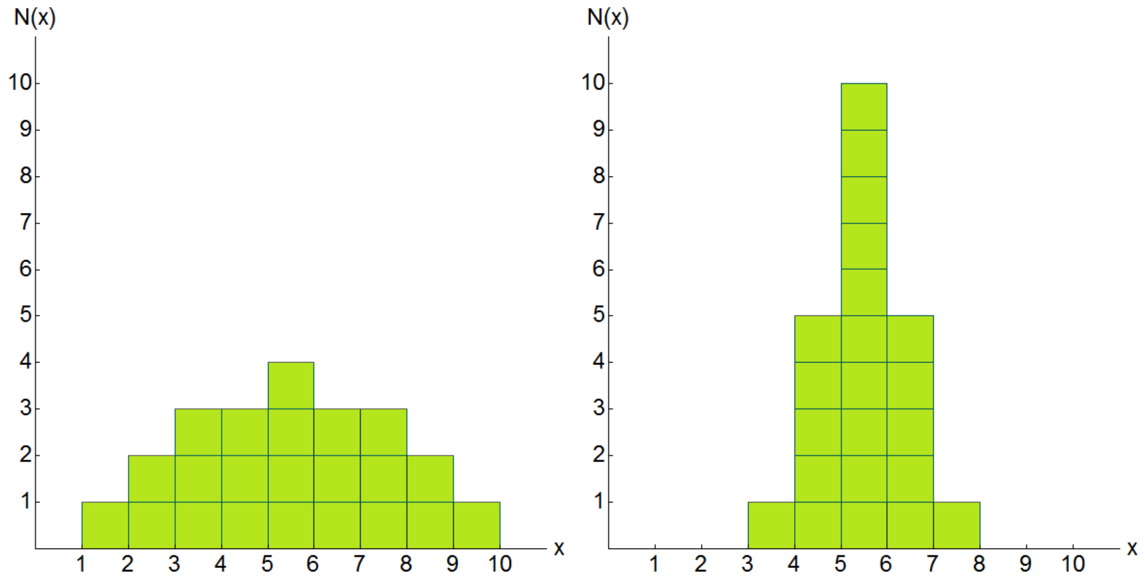


Figure 1.5: *Two histograms outlining the differences in how data is distributed. The mean, median, and most probable value are the same for both distributions. Despite this, the plots are not identical.*

The histograms above reveals that the "spread" of the data is yet another measurable quantity. One might think that this can be estimated by taking the average distance from the mean value. However, due to cancelation of negative and positive terms, this will ultimately lead to zero. Instead, the square of each distance from the mean is calculated and the average of these values is used. This is known as the *variance*. For the set $X = \{x_1, x_2 \ldots x_n\}$ the variance can be expressed mathematically as

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N} (x_n - \bar{X})^2 \,. \tag{1.7}$$

Once again, N is the length of the data set and $\bar{X}$ is the mean of the set. Variance is a fundamental statistical measurement, and it plays a large role in noise analysis. The reason it is defined as σ^2 is because it is often written in terms of *standard deviation*, which is defined as:

$$\sigma = \sqrt{\sigma^2}\,. \tag{1.8}$$

A set of data whose points are more likely to be found close to the mean rather than farther are said to be *normally distributed*. A subsequent histogram of such a data set will be bell shaped, or *Gaussian* (Figure 1.6).

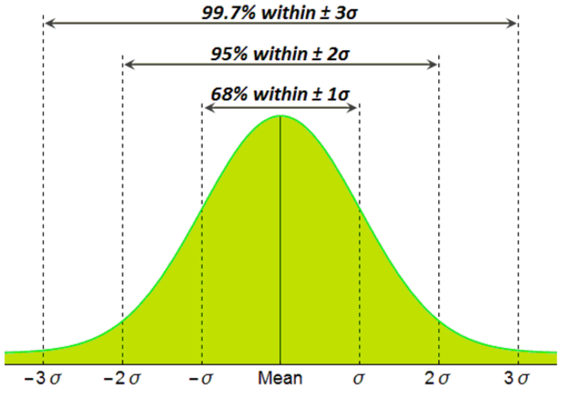


***Figure 1.6**: A Gaussian distribution function. Note that nearly all the data points lay within 3 standard deviations of the data.*

1.5.3 Covariance and Correlation

It is also useful to investigate how dependent two or more variables are on each other. To get a sense of this, one can define the *covariance* between sets of data. It shows how much one variable changes with another, if at all, and can be used to make predictions about trends in data. For two sets of discrete data $X = \{x_1, x_2 \ldots x_n\}$ and $Y = \{y_1, y_2 \ldots y_n\}$, both of length N, the covariance function can be approximated by the sample covariance given by[10]:

$$cov_{XY} \approx r_{XY} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{X})(y_n - \bar{Y})\ . \tag{1.9}$$

The same analysis can be done on a given set of data with itself. This can be used to see if data entries in a list depend on previous values in the same sequence. This is referred to as the *autocovariance* of the data, and is once again approximated by[10]:

$$cov_{XX} \approx r(\tau) = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{X})(x_{n+\tau} - \bar{X})\ . \tag{1.10}$$

The iterator τ in the formula above represents the lag, or spacing, between each data point. τ can be expressed as sample n multiplied with the time interval between samples, τ_0, so that $\tau = n\tau_0$. It is possible to have a lag of 0 between points. This means that each point's distance from the mean is squared and the average of these values is taken. In other words, the autocovariance at lag 0 reduces to the variance of the data:

$$r(0) = \sigma_X^2. \tag{1.11}$$

Given that it measures the dependence of values in a set of data, the autocovariance can also be thought of as the signal's "memory". For white noise, each value is independent of the previous entry in the set. Thus, the autocovariance at any lag time other than 0 is zero. This is expressed as:

$$r(\tau) = \sigma_X^2 \cdot \delta_{tt'} \ . \tag{1.12}$$

Here, $\delta_{tt'}$ is the Kronecker delta function. It is equal to 1 for time t equal to time t' (zero lag), and is equal to 0 for all times t not equal to time t'. If there are *N* data points, then there exists *N-1* lags between all the points. Therefore, τ is indexed from *0* to *N-1*. The calculated coefficient associated with each covariance can be a large or small number depending on the data. Some data sets may have larger coefficients, yet be less correlated than others. It is; therefore, more practical to normalize the covariance by dividing out the standard deviation of each set of data. When this is done for two different variables, the result is known as the *cross-correlation*, or simply *correlation*. And when the data is correlated with itself, it is called *auto-correlation*[10]:

$$corr_{XX} \approx R(\tau) = \frac{r(\tau)}{\sigma_X^2} = \frac{1}{N\sigma_X^2}\sum_{n=1}^{N}(x_n - \bar{X})(x_{n+\tau} - \bar{X}) \ . \tag{1.13}$$

1.5.4 Stationarity and Ergodicity

The statistical definitions given above can be used to describe any finite set of discrete data. When the mean, variance, and correlation functions of a given list do not change in time, then the process is said to be *stationary*. It can also be the case that the statistical properties of a single set of data will closely resemble the same properties of an average over an ensemble of realizations produced by the same process. In this case, the process is also *ergodic*.

Chapter 2

Spectral Analysis

Examination of the time domain can be useful for obtaining certain information about a signal. For instance, it might be apparent that the signal has a particular waveform. However, this type of analysis might not be straightforward for a mixture of signals, or for a signal affected by noise. For these, it is more useful to examine the frequency spectrum of the signal. A very powerful tool in mathematics is the ability to decompose any signal's waveform into a sum of sine and cosine functions. This is accomplished through the *Fourier Transform*. The Fourier Transform takes data from the time domain and maps it to the frequency domain, thereby specifying how much of a given frequency the signal is composed of. An example of its application is illustrated in Figure 2.1.

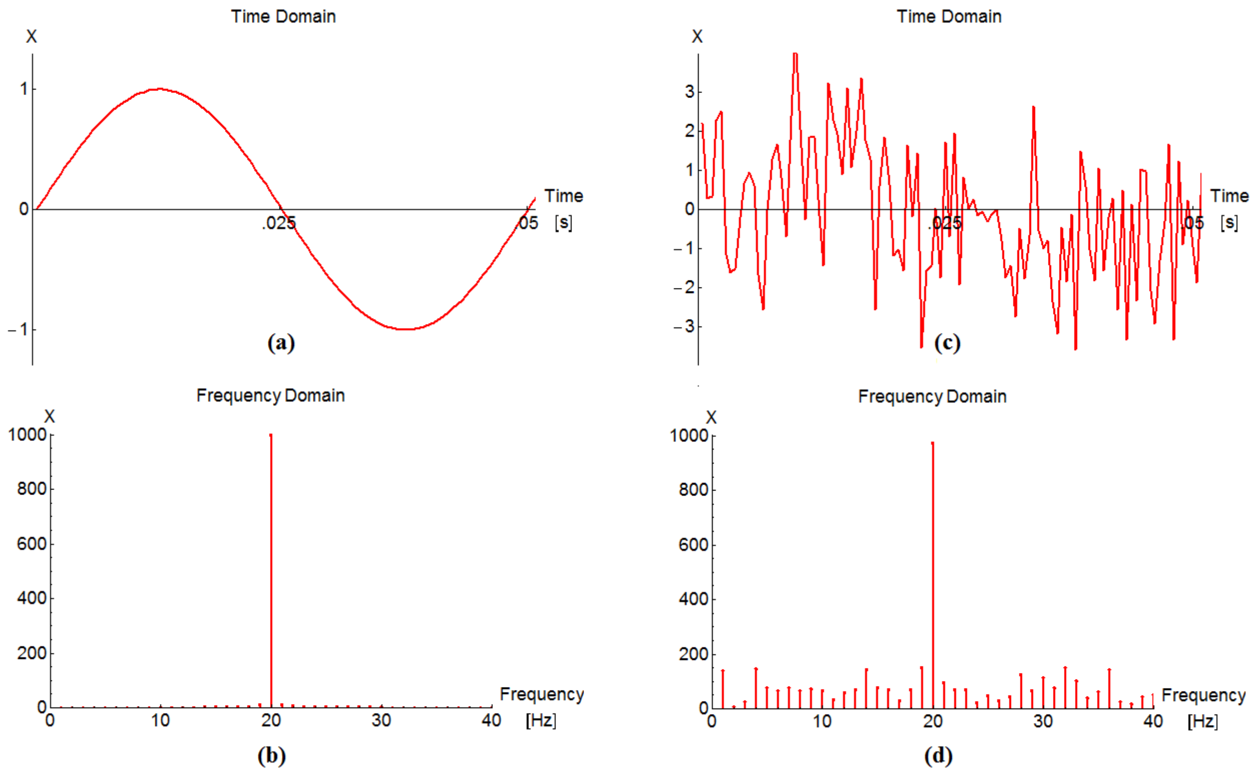


Figure 2.1*: An application of the Fourier transform. (a) A 20 Hz sine wave. (b) the Fourier transform of the sine wave. 20 Hz is the only frequency present. (c) The same 20 Hz signal, but with added Gaussian white noise. (d) The Fourier transform of the noisy signal. Note that the 20 Hz sine wave still dominates the frequency spectrum.*

2.1 Fourier Transforms

There are several ways to define the Fourier Transform that each correspond to the type of signal being analyzed. Unfortunately, sines and cosines extend from -∞ to +∞, and so using them to describe a finite signal requires one of two conditions. The signal itself can be said to extend to plus and minus infinity with all non-measured values set equal to zero, making it aperiodic. Or, the signal can span the same range and be imagined to repeat itself, thus becoming periodic. Furthermore, both cases can be applied to continuous and discrete data sets, making for a total of four types of transforms[11]. An example of each has been outlined below in Figure 2.2.

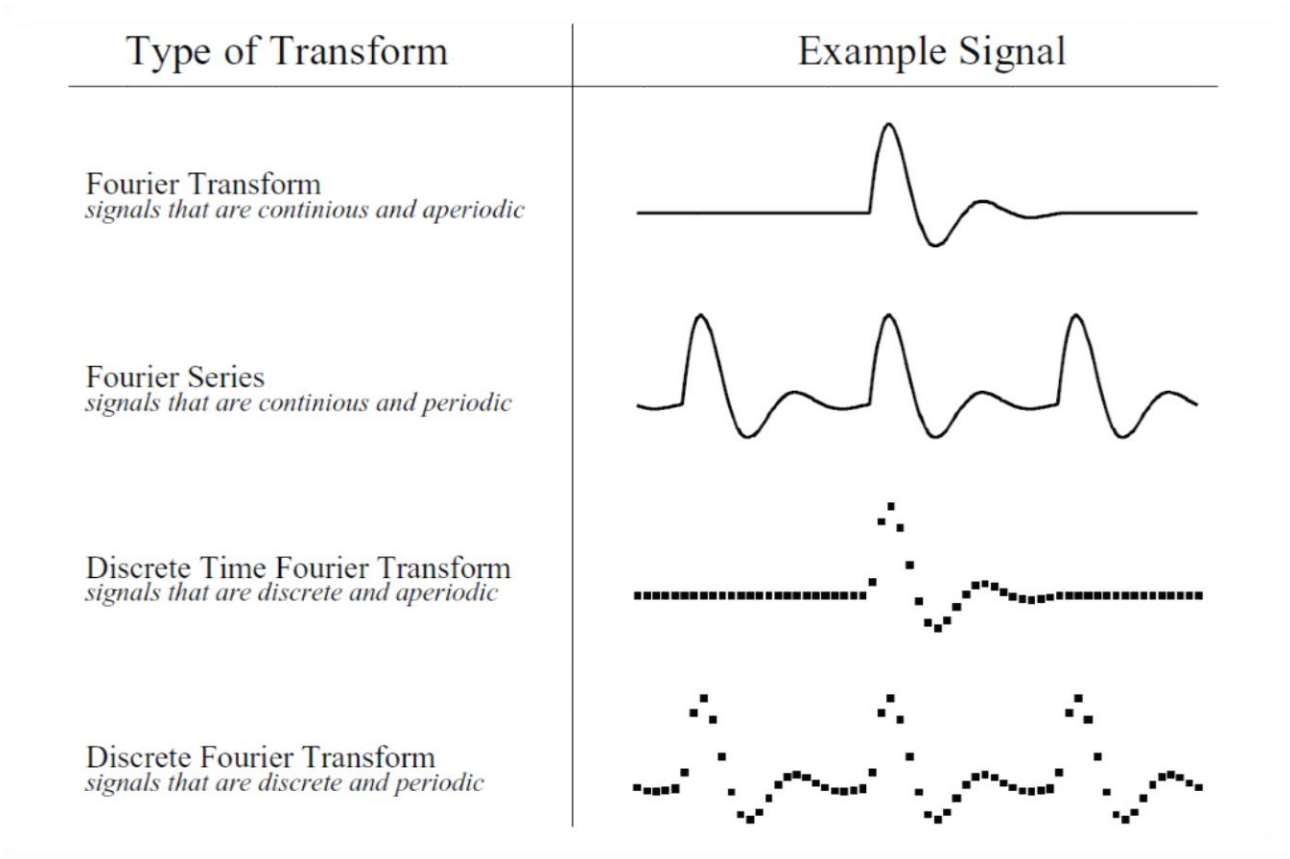

Type of Transform
Example Signal
Fourier Transform
signals that are continious and aperiodic
Fourier Series
signals that are continious and periodic
Discrete Time Fourier Transform
signals that are discrete and aperiodic
Discrete Fourier Transform
signals that are discrete and periodic

Figure 2.2*: Example signals associated with each Fourier transform*[11].

Conversely, each transform has the ability to change frequency data back into the time domain. This is done using the *Inverse Fourier Transform*. The formulas corresponding to the transforms outlined above and their inverses are expressed mathematically as follows:

Fourier Transform (FT):

$$X(\omega) = \int_{-\infty}^{\infty} x(t) \cdot e^{-i\omega t} \cdot dt \quad \leftrightarrow \quad x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) \cdot e^{i\omega t} \cdot d\omega \tag{2.1}$$

Fourier Series (FS):

$$a_k = \frac{1}{T} \int_{0}^{T} x(t) \cdot e^{-ik\omega_0 t} \cdot dt \quad \leftrightarrow \quad x(t) = \sum_{k=-\infty}^{\infty} a_k \cdot e^{ik\omega_0 t} \tag{2.2}$$

Discrete Time Fourier Transform (DTFT):

$$X(k) = \sum_{n=-\infty}^{\infty} x(n) \cdot e^{-ikt} \quad \leftrightarrow \quad x(t) = \frac{1}{N} \sum_{k} X(k) \cdot e^{ikt} \tag{2.3}$$

If the DTFT is sampled over a finite time interval, T, then it can be imagined that the sampled segment is repeated from -∞ to +∞. Since it is now periodic over the sampling period T, then $x(t) = x(t + T)$. This limits the frequency ω to being integer multiples of ω_0, or, using the definition for the fundamental frequency given by Equation (1.3): $\omega_k \rightarrow k\omega_0 = k2\pi/T$. Also, the period of measurement is given by the number of points multiplied by the interval of time between them, $T \rightarrow N\tau_0$. Furthermore, the time that has elapsed, t, now becomes the time that has elapsed to reach the n^{th} point under consideration : $t_n \rightarrow n\tau_0$. Putting all of the pieces together yields the *Discrete Fourier Transform*[12].

Discrete Fourier Transform (DFT):

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi kn/N} \quad \leftrightarrow \quad x(n) = \frac{1}{N} \sum_{k=0}^{N-1} a_k \cdot e^{i2\pi kn/N} \tag{2.4}$$

The number of frequency bins, *k*, is equal to the number of samples being processed, *N*. A typical DTFT frequency spectrum will range from zero to half of the frequency with which the data was sampled. After this point, *aliasing* occurs. Aliasing is the allocation of a signal's strength to frequencies that it isn't actually composed of. Visually, this has the effect of reflecting the frequency spectrum about its midpoint. The cutoff frequency after which aliasing occurs is known as the *Nyquist frequency*. This can be applied to the DFT as well. If the signal is real, then $X(-k) = X(k)^*$. This means that all of the *N* data points become spread out between *– N/2* and *N/2*, and they are mirrored about zero. This implies that the complete frequency spectrum can be described by frequencies ranging from 0 to *N/2*. Thus, the Nyquist frequency of the DFT is defined as *N/2*.

Even with the examples outlined in Figure 2.2, it can still be unclear which transform to use for a finite set of data. One would be tempted to use the DTFT, given that it is meant for a non-periodic signal, and a random set of data may or may not be periodic. However, the frequency spectrum of the DTFT is continuous, and the transform is defined for the range -∞ to +∞. It is unlikely that an infinite set of data will be under consideration. On the other hand, the DFT is defined over a finite range, meaning a finite set of data. It is worth noting that the literature on the subject usually uses the DTFT in theory and proofs. This is more correct for theory, as the DFT yields the frequency spectrum of the "imagined" periodic signal, sampled over a finite interval. Assuming that the function is periodic; however, is often disregarded in lieu of computational processing. The reason for this is that computing the inverse DFT will result in recovering the original signal back (Figure 2.3), and so assuming periodicity has little consequence.

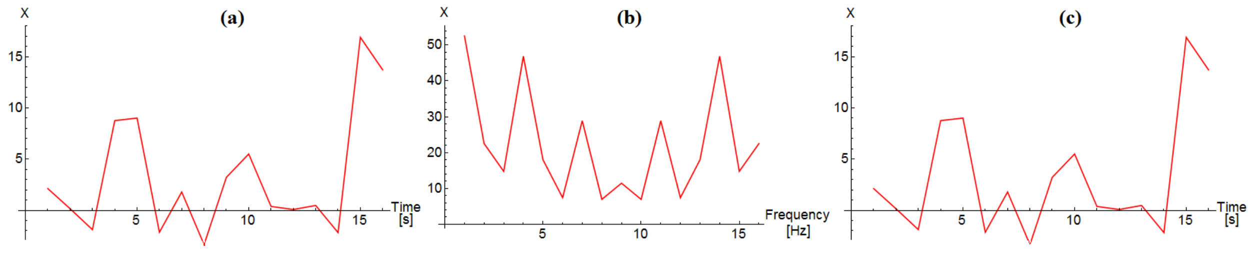


Figure 2.3*: The result of performing the inverse DFT on the DFT of a time series. (a) Untransformed time series data. (b) The DFT of the time series. (c) The inverse DFT of the DFT. The original time series is recovered.*

2.2 Energy of a Signal

The total energy of a discrete signal is conventionally defined as[12]

$$E_x = \sum_{n=1}^{N} |x(n)|^2 \,, \tag{2.5}$$

where *x(n)* is the nth value in the time domain. Parseval's theorem states that the energy of a waveform is the same whether measuring it from the time domain or the frequency domain[13]. For discrete signals, the proof is as follows: Suppose there is a list with *N* values of *x(n)*, and that *X(k)* is related to *x(n)* via a Discrete Fourier Transform (Equation 2.4):

$$X(k) = \sum_{n=1}^{N} x(n) \cdot e^{-i2\pi nk/N} \,,$$

where *n* and *k* are, once again, the current sample number and frequency under consideration. Squaring the left side and multiplying the right by its complex conjugate yields

$$|X(k)|^2 = \sum_{n=1}^{N} x(n) \sum_{n\prime=1}^{N} x^*(n') e^{i2\pi(n-n')k/N}. \tag{2.6}$$

Summing both sides gives the total energy of the signal

$$\sum_{k=1}^{N} |X(k)|^2 = \sum_{k=1}^{N}\sum_{n=1}^{N} x(n) \sum_{n\prime=1}^{N} x^*(n') e^{\frac{i2\pi(n-n')k}{N}} \tag{2.7}$$

$$= \sum_{n=1}^{N} x(n) \sum_{n\prime=1}^{N} x^*(n') \sum_{k=1}^{N} e^{i2\pi(n-n')k/N} . \tag{2.8}$$

The final sum in this expression is a geometric series, which can be calculated as

$$\sum_{k=1}^{N} e^{-i2\pi(n-n')k/N} = \frac{e^{i2\pi(n-n')} - 1}{e^{i2\pi(n-n')/N} - 1}. \tag{2.9}$$

The right hand side is evaluated as *0* when $n \neq n'$ and is N when $n = n'$. Therefore

$$\sum_{k=1}^{N} e^{-i2\pi(n-n')k/N} = N\delta_{nn\prime} . \tag{2.10}$$

Consequently

$$E_x = \frac{1}{N}\sum_{k=1}^{N} |X(k)|^2 = \sum_{n=1}^{N} |x(n)|^2 . \tag{2.11}$$

The summation in this case is performed over a finite range of data. Doing this for an infinite-length signal would theoretically result in infinite energy. Given that the total energy is represented by the average of each frequency component squared, the energy density, or energy per frequency, is expressed as:

$$ESD = |X(k)|^2. \tag{2.12}$$

Note that the signal's energy might not be described in the physical sense of the term. Depending on what is actually being measured, the units of energy may not work out to be entirely what one would expect. To get the units to work out then, the Fourier transform can be scaled by the time step, Δt, that separates each data point. Recall that for an evenly sampled signal, the duration of time between the points is τ_0. This rescales the energy spectrum to:

$$ESD = \tau_0^2 \cdot |X(k)|^2. \tag{2.13}$$

2.3 The Power Spectrum

Power is defined as the energy per unit of time. The duration of time the signal encompasses is, again, the number of data samples multiplied by the time between each point, $T = N \cdot \tau_0$. Dividing this into the energy spectrum yields the power spectral density (PSD)[12]:

$$PSD = S(k) = \frac{\tau_0}{N} \cdot |X(k)|^2. \tag{2.14}$$

This manner of power spectral estimation is referred to as the periodogram. A drawback of using this method is that the power spectrum does not converge to its true value regardless of how many samples it is computed over. An example of this is illustrated in Figure 2.4.

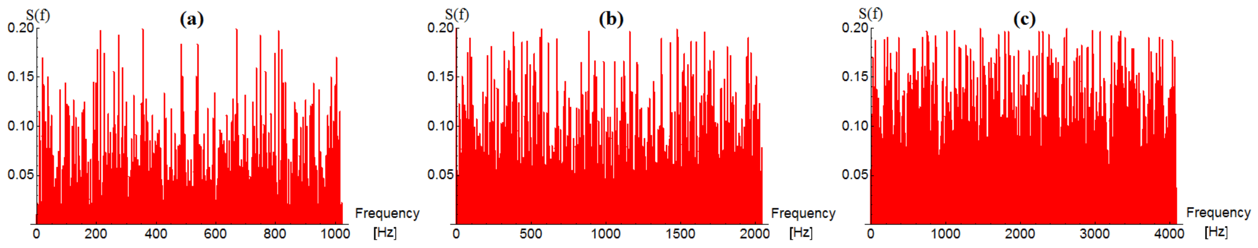


***Figure 2.4**: Example of variance in the periodogram. (a) Power spectrum for 1000 samples. (b) Power spectrum for 2000 samples. (c) Power spectrum for 4000 samples.*

In the example above, the number of data points sampled begins at 1000, and is then doubled twice. The average of each spectrum is the same at a value of .048; however, the variance of each one also has an identical value of .002. The variance of the spectrum did not decrease with the number of samples. There are alternative methods for estimating the power spectrum. Due to time constraints, the periodogram remains the method chosen for this research. The lack of convergence has been dealt with by averaging over an ensemble of power spectra in chapter 4.

2.3.1 Wiener–Khinchin Theorem

One way to check that the correct power spectrum has been attained is to take the DFT of the autocovariance function described earlier. This principal is known as the *Weiner-Khinchin Theorem*, and it states that the PSD and autocovariance are Fourier transform pairs[13]:

$$S(f) = \int_{-\infty}^{\infty} r(\tau) \cdot e^{-i2\pi f t} \cdot d\tau \,. \tag{2.15}$$

To approximate this for a discrete signal, the integral is replaced by a sum, and the exponential takes the same form as in the DFT. The time step $d\tau$ becomes the time interval in between data points, τ_0.

$$S(f) = \tau_0 \cdot \sum_{n=-\infty}^{N} r(n) \cdot e^{-i2\pi fn/N} \,. \tag{2.16}$$

Note the change from $r(\tau)$ to $r(n)$, this is because the nth part of the covariance function is now under consideration in the transform. Recall that the autocovariance of white noise is equal to its variance multiplied with the Kronecker delta (Equation 1.12). Modifying the index of the Kronecker from time, t, to list position, n, and plugging it into Equation (2.16) yields:

$$S(f) = \tau_0 \cdot \sum_{n=-\infty}^{N} \sigma_X^2 \cdot \delta_{nn'} \cdot e^{-i2\pi fn/N} = \tau_0 \cdot \sigma_X^2 \,. \tag{2.17}$$

This means that white noise will have a flat power spectrum held at a constant value of $\tau_0 \cdot \sigma_X^2$. In the example shown in Figure 2.4, τ_0 was set to $1/1024$ seconds, and the variance of the data was taken to be 49. Multiplying these numbers together yields a value of .04785, which is precisely what the power spectrum of that signal averages to.

Chapter 3

Noise Characterization

In general, any periodic signal can be described as having both a phase and amplitude. Instabilities in either of these components leads to instantaneous values differing from their actual value. These discrepancies are what is referred to as the *noise* of the data[14].

3.1 Frequency Stability

Oscillators often undergo what is known as *frequency drift*. This means that their operating frequency will change with time due to a variety of factors such as aging of the oscillator, or outside environmental factors. Drift typically has units of Hz/s, and can; therefore, be thought of as velocity in a sense. On the other hand, *frequency stability* describes how little a clock is prone to frequency drift. Consider, once again, the function *Y(t)* defined earlier, now with the following modifications:

$$Y(t) = [Y_0 + \varepsilon(t)] \cdot \sin[2\pi\nu_0 t + \varphi(t)]. \tag{3.1}$$

Where,	Y_0	= Nominal amplitude
	$\varepsilon(t)$	= Deviation from nominal amplitude
	ν_0	= Nominal frequency
	$\varphi(t)$	= Deviation from nominal frequency

Both $\varepsilon(t)$ and $\varphi(t)$ take the form of added noise. The phase of the signal is defined by the argument of *sine*. The fluctuations defined by $\varphi(t)$ are phase modulated (PM) noise[15]. Taking the derivative of this argument will give the *instantaneous angular frequency, ω(t)*:

$$\omega(t) = \frac{d}{dt}[2\pi\nu_0 t + \varphi(t)] = 2\pi\nu_0 + \frac{d\varphi}{dt}, \tag{3.2}$$

which can be expressed in Hz by dividing out 2π:

$$\nu(t) = \nu_0 + \frac{1}{2\pi}\frac{d\varphi}{dt}. \tag{3.3}$$

Fluctuations in this form are frequency modulated (FM) noise[15].

3.2 Noise Power Laws

Five types of noise have been identified through their distinctive power spectra. The power spectrum of a given noise type will follow the relationship[10]

$$S(f) = h_\alpha \cdot f^\alpha \,. \tag{3.4}$$

This states that the power of a signal will be proportional to a given frequency, f, raised to some power α. Each one is classified below, and has its own contribution (h_α) to a given signal.

α	Noise Name	Corresponding Figure
-2	Random Walk FM	Figure 3.1
-1	Flicker FM	Figure 3.2
0	White FM	Figure 3.3
1	Flicker PM	Figure 3.4
2	White PM	Figure 3.5

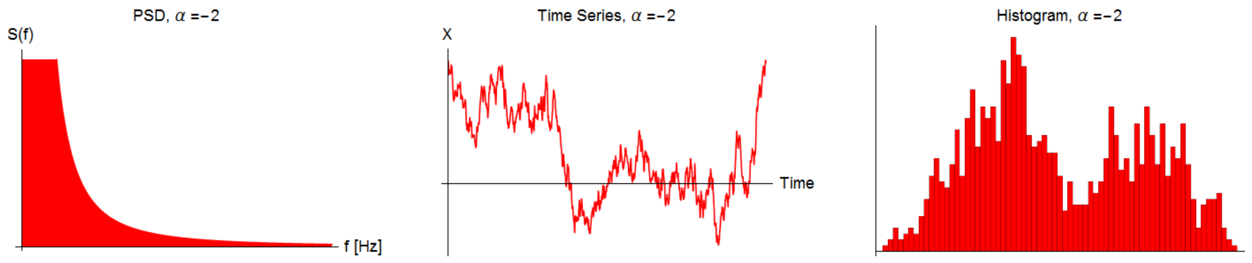


Figure 3.1: *The power spectrum, time series, and histogram of random walk FM noise.*

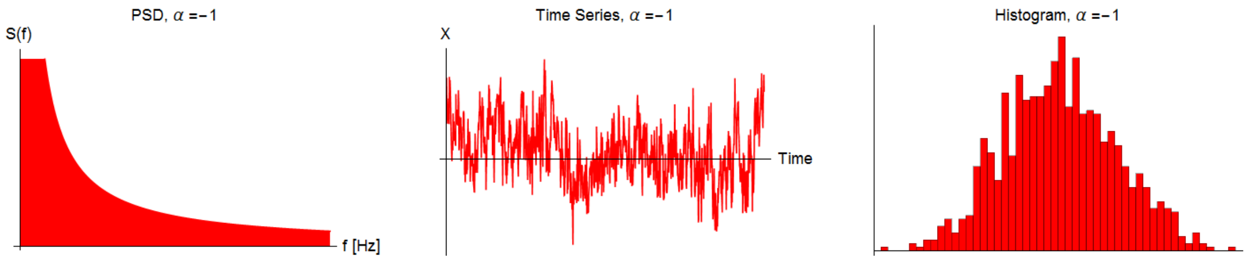


Figure 3.2: *The power spectrum, time series, and histogram of flicker FM noise.*

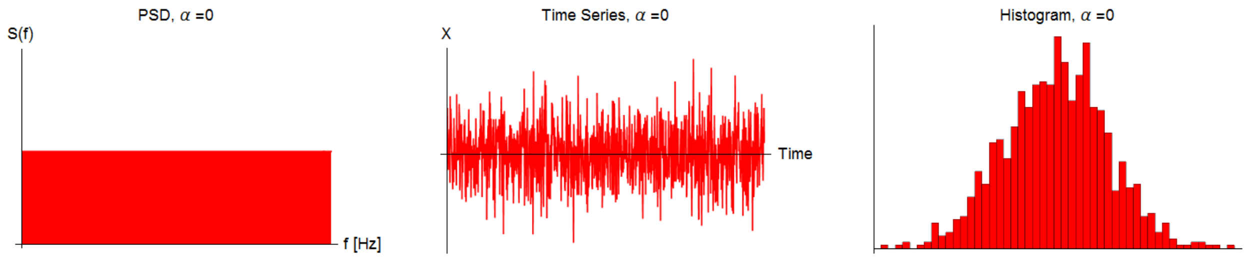


Figure 3.3: *The power spectrum, time series, and histogram of white FM noise.*

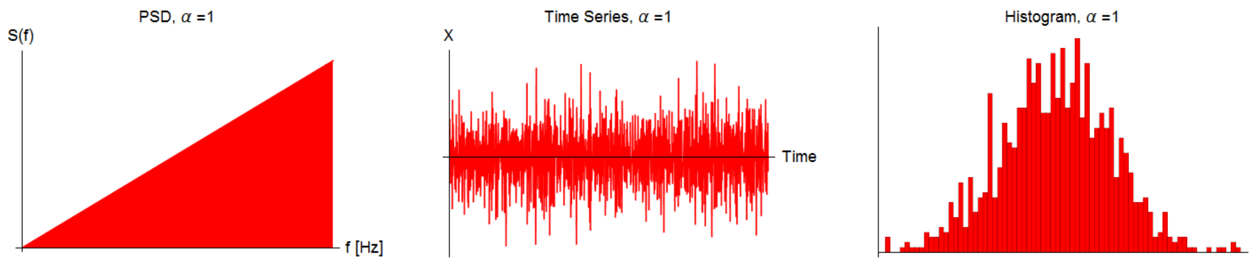


***Figure 3.4**: The power spectrum, time series, and histogram of flicker PM noise.*

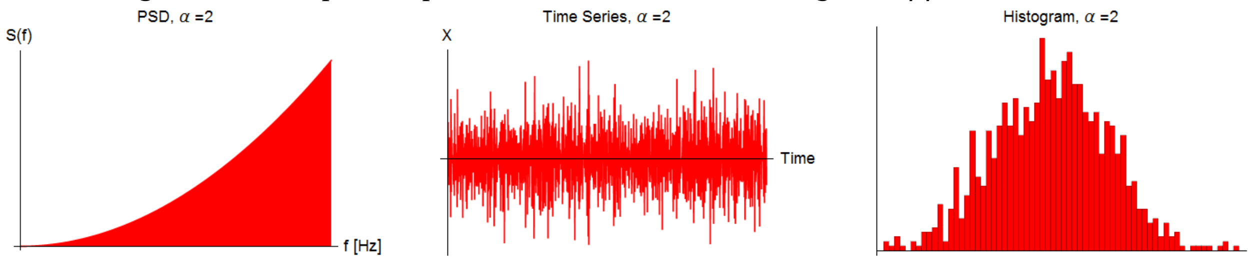


***Figure 3.5**: The power spectrum, time series, and histogram of white PM noise.*

The reason for including histograms is to get a sense of the statistical properties associated with each type of noise. For the most part, each follows a Gaussian distribution. However, random walk breaks this pattern in that it does not have a clear distribution. In addition to this, each realization of random walk will generate a different histogram altogether. Thus, the random walk is considered a non-stationary process. Because it is non-stationary, things like the autocovariance function do not technically exist for it. Random walk is described as noise that has been added onto the previous entry in a list. One can visualize random walk in the analogy of a person walking in random directions. Each step is limited by the length of the person's legs, and the person's current position depends on where they were one step ago. Random walk exhibits a strong autocorrelation. In order to apply some of the processing tools developed then, it must be transformed into white noise. This is achieved by taking the difference between data points in the random walk. Mathematically, random walk can be expressed as:

$$y_t = y_{t-1} + \varepsilon_t, \tag{3.5}$$

Where the current value, y_t, is equal to the previous value, y_{t-1}, with added Gaussian white noise, ε_t. Following this definition, the next term in the series will be:

$$y_{t+1} = y_t + \varepsilon_{t'}. \tag{3.6}$$

Differencing any two adjacent points in the list will yield:

$$\begin{aligned} y_{t+1} - y_t &= (y_t + \varepsilon_{t'}) - (y_{t-1} + \varepsilon_t) \\ &= (y_{t-1} + \varepsilon_t + \varepsilon_{t'}) - (y_{t-1} + \varepsilon_t) \\ &= \varepsilon_{t'} \end{aligned} \tag{3.7}$$

Thus, it is shown that differenced random walk produces Gaussian white noise.

Flicker FM noise resembles a mixture of random walk and white FM noise. For the most part, its statistical properties do not change. For this reason, it is referred to as being weakly stationary. It too can be differenced into white noise.

A considerable amount of examples used so far have pertained to flat spectrum (white) noise. This type of noise will be more closely analyzed in chapter 4. Oscillators within various types of clocks can be affected by all of the power law noise described above. However, in the case of atomic clocks, Rubidium and Cesium oscillators exhibit noise mostly in the range of $\alpha = -2, -1, 0$ (Figure 3.6). These correspond to the random walk, flicker FM, and white FM noise types respectively[16].

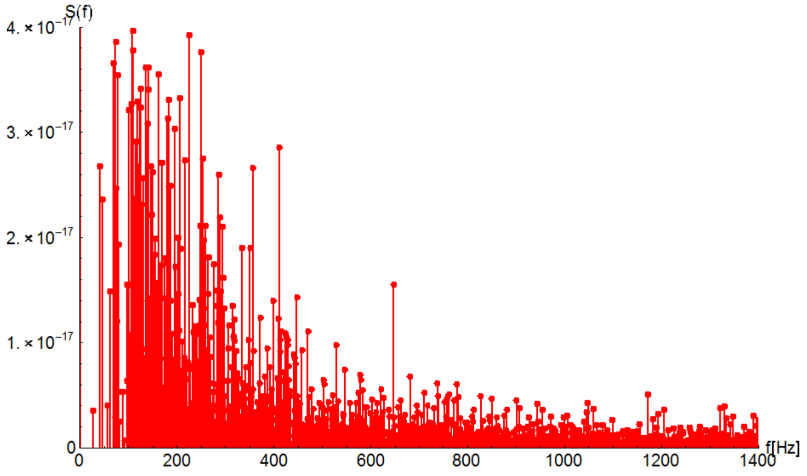


***Figure 3.6**: The power spectrum of the clock bias shown in Figure 1.3.*

3.3 Generating a New Time Series from the Given PSD

Once the power spectrum of a signal has been defined, it is possible to then generate a time series with the same statistical properties that will produce the same spectrum. The procedure for applying this numerically is outlined in chapter 4, but it is shown analytically here. Suppose there is a target power spectrum denoted by $S(f)_{target}$. This power spectrum will have been produced by taking the square magnitude of the DFT of a time series, per Equation (2.14). This random time series will be denoted by $X(t)_{target}$. The DFT of this series gives back a list of complex terms. This list will be called $X(f)_{target}$. The phase information contained in each complex value is lost when the magnitude of the DFT is computed. Therefore, new phase information must be generated if $X(t)_{target}$ is to be recovered from the power spectrum. This phase information can come from the DFT of a newly generated random list with the same number of data entries as $X(t)_{target}$. The DFT of this new list will be denoted as $Y(f)_{new}$. Dividing the DFT by its conjugate will normalize it, leaving only the phase information left. Now that the phase information has been created, the proper magnitude of the transform will need to be evaluated. It can be easily obtained by taking the square root of $S(f)_{target}$. This will be equal to the magnitude of the frequency spectrum since

$$\sqrt{S(f)_{Target}} \propto \left|X(f)_{Target}\right| . \tag{3.8}$$

Multiplying *Y(f)_{new}* with $\sqrt{S(f)_{Target}}$ now yields the appropriate frequency spectrum complete with phase information. Taking the inverse DFT of this spectrum will reproduce a signal in the time domain, denoted by $Z(t)_{Reconstructed}$. This new signal will not be identical to $X(t)_{target}$. However, it does have the same statistical properties as $X(t)_{target}$.

It is straight forward to check that this new signal will reproduce the same power spectrum as before. The DFT of the reconstructed signal is:

$$Z(f)_{Reconstructed} = \frac{Y(f)_{new}}{Y(f)^*_{new}} \cdot \sqrt{S(f)_{Target}} \tag{3.9}$$

Taking the square magnitude of this, as per Equation (2.14), yields:

$$|Z(f)_{Reconstructed}|^2 = \left(\frac{Y(f)_{new}}{Y(f)^*_{new}} \cdot \frac{Y(f)^*_{new}}{Y(f)_{new}} \cdot \sqrt{S(f)_{Target}}\right)^2 \tag{3.10}$$

$$= \sqrt{S(f)_{Target}}^2$$

$$= S(f)_{Target}$$

This shows that the power spectrum of the reconstructed signal is equal to the power spectrum of the target signal. Note that there was no need to renormalized the spectrum by a factor of $\frac{\tau_0}{N}$ as in Equation (2.14) because this value has already been accounted for in $S(f)_{target}$.

Chapter 4

Implementation in Mathematica

Mathematica is a mathematical computation program developed by Wolfram Research of Champaign, Illinois. It contains a multitude of function libraries which can be called in the following way `FunctionName[FunctionArguement]`. Mathematica has built in algorithms for processing large amounts of data in a short period of time. It is also relatively simple to use, and there are a multitude of help pages and example codes available online. For these reasons, it has been selected as the program used to model the noise simulation.

4.1 Generating a Random Signal

The simplest form of noise to generate is Gaussian White noise. Mathematica comes with a built in random number generator, `RandomReal[]`, utilized to create a list of randomized discrete data of specified length. The desired standard deviation, or "spread", of the data can be defined as well. Furthermore, this can be combined with the `NormalDistribution[]` function to produce Gaussian data centered about a specified mean value. Figure 4.1 illustrates the result of combining these functions.

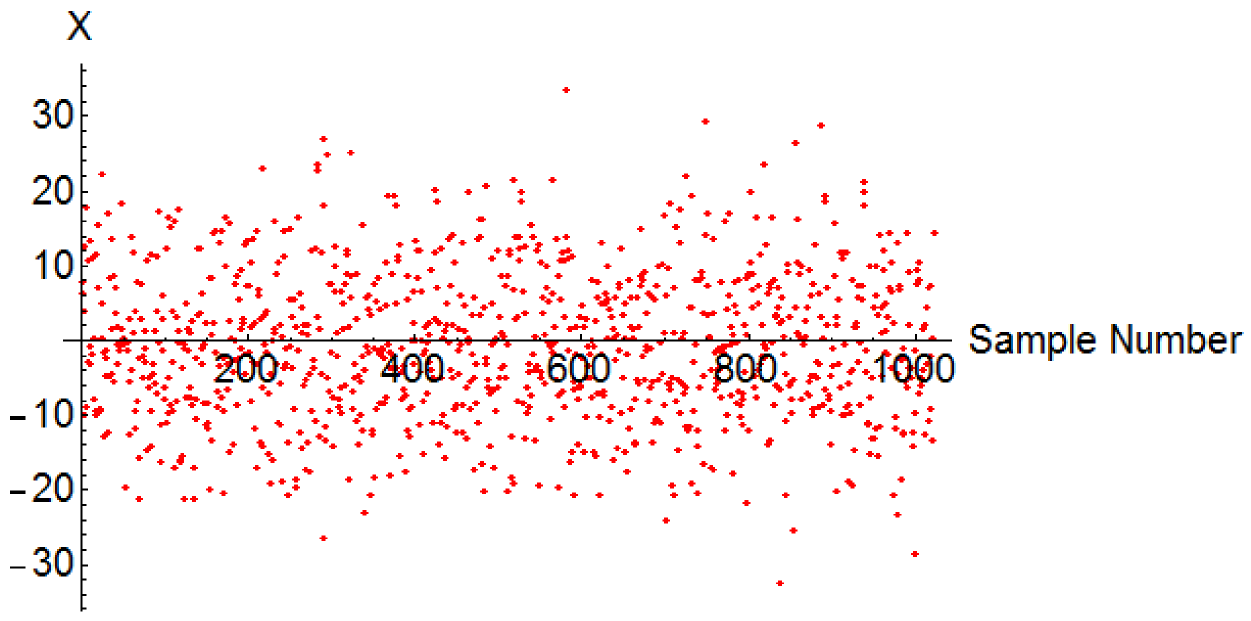


***Figure 4.1**: A signal composed primarily of Gaussian white noise.*

The image shown above was produced by calling Mathematica's `ListPlot[]` command. There are 2^{10}, or 1024, random data points contained within the list. For programming in general, more efficient computations can be carried out when a given set of data has a length equal to a power of 2. It can be imagined that the data produced is the result of sampling a real, continuous signal at regular intervals. The duration of time that the signal is sampled for is arbitrarily chosen to be 1 second. This makes the sampling frequency 1024 Hz. And so, a Nyquist frequency of 512 Hz will be expected. The data has also been prescribed an arbitrary standard deviation of 7, and is normally distributed about 0. A histogram of this data, shown in Figure 4.2, reflects these statistical properties.

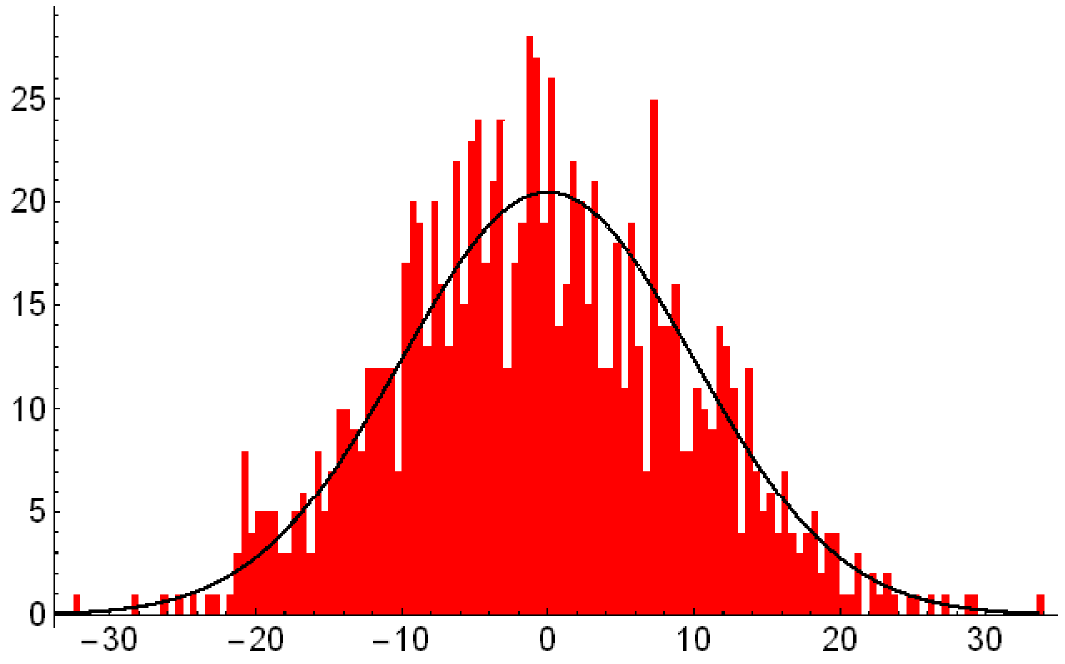


Figure 4.2*: A histogram of the generated white noise.*

Although the mean and standard deviation have been specified, the random list will not exhibit these values identically. For the random signal generated above, the calculated mean is .33, while the calculated standard deviation is 6.88.

Next, the frequency spectrum of the generated noise is examined. For this, the function `Fourier[]` has been called to compute the DFT of the signal. One must be careful when using this function, as Mathematica defines it in the following fashion:

$$X(k) = \frac{1}{N^{(1-a)/2}} \sum_{n=1}^{N} x(n) \cdot e^{i2\pi b(k-1)(n-1)/N}, \qquad (4.1)$$

where the terms ***a*** and ***b*** can be set using FourierParameters→{a,b}. The iterator ***n*** starts at 1 because it refers to the position in the list being transformed, and there is no 0th position. ***N*** is the total number of data points, and ***k*** is the frequency under consideration. To get the desired transform given by Equation (4.1), ***a*** and ***b*** have been set to 1 and -1 respectively. When taking a transform numerically, one gets back a list of complex terms. Each frequency's contribution to the signal can be expressed by the magnitude of these complex terms. In Mathematica, this is accomplished by either calling the absolute value, `Abs[]`, of the Fourier transform, or by multiplying the transformed data by its conjugate, `Conjugate[]`, and taking the square root, `Sqrt[]`. The result of doing this is captured in Figure 4.3.

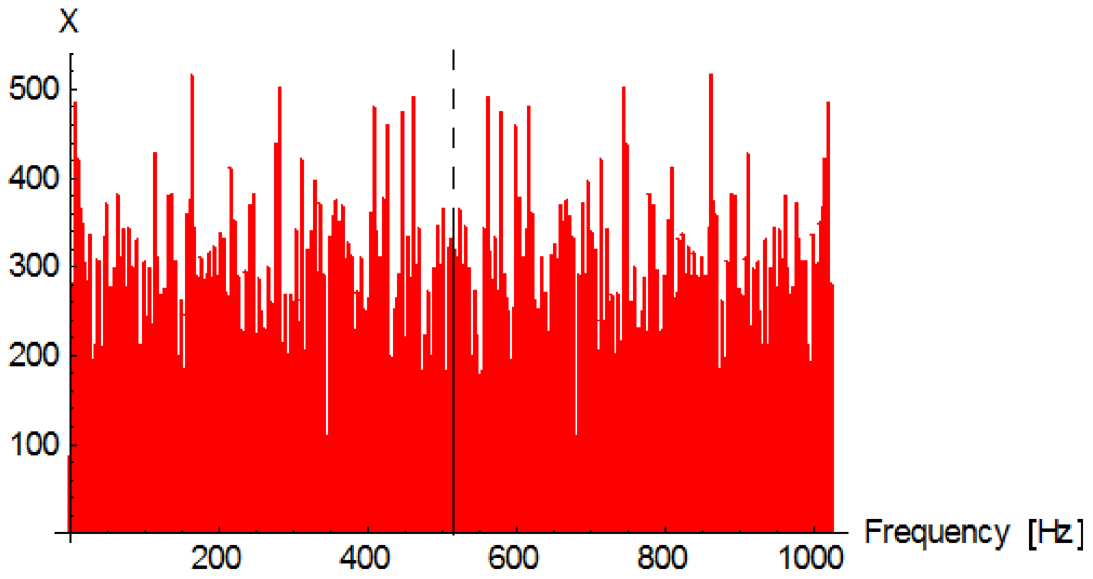


***Figure 4.3**: The frequency spectrum of the generated white noise.*

As expected, the white noise signal is an amalgamation of many frequencies, each with a similar contribution to the signal's energy. Note that the signal is mirrored about the Nyquist frequency of 512 Hz as a result of aliasing. Parseval's Theorem, described by Equation (2.11), can be used to check that the energies in each domain are equal. Applying it to the time domain of this data set yields a value of 47,643. Taking the square magnitude of the frequency spectrum and dividing it by the number of data points also yields a value of 47,643. Although difficult to discern from the graph, it is important to point out that both the zero frequency and Nyquist frequency only appear once in the Fourier transform. This will be applied later, when reconstructing a signal from a power spectrum.

Equation (2.14) is then applied to the data to estimate the power spectrum of the random signal. Again, white noise has equal contributions from many frequencies, and so one would expect the power spectrum to be flat. However, Figure 4.4 shows that this is not quite the case.

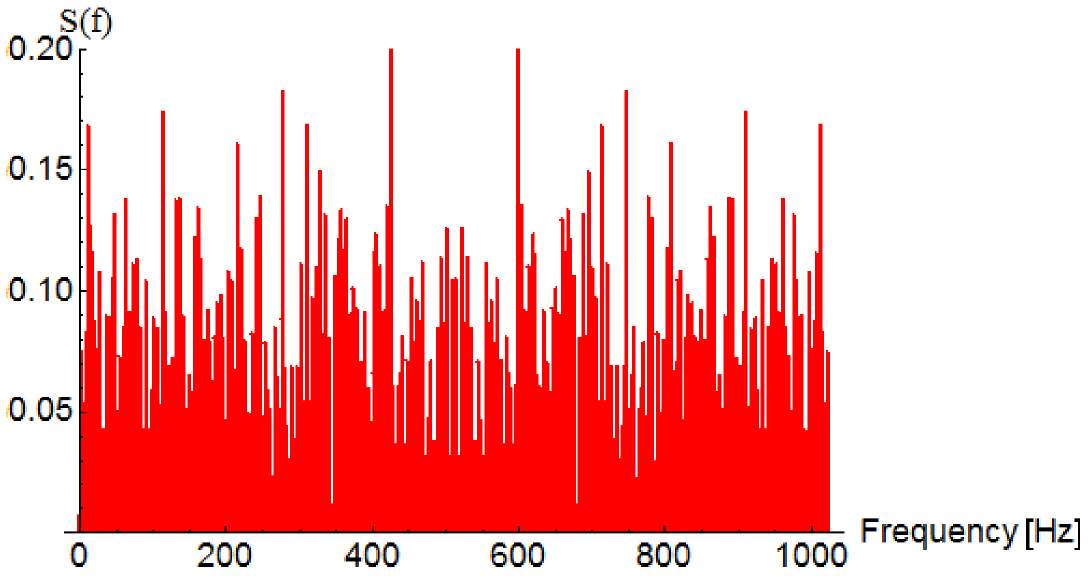


Figure 4.4*: The power spectrum of the generated white noise.*

The power spectrum has fluctuations because this is a single realization of data under consideration. Because white noise is ergodic, the situation can be remedied by taking the average over an ensemble of power spectra generated from white noise with the same statistical parameters. Implementing this demonstrates how the power spectrum converges to its true value for the given noise.

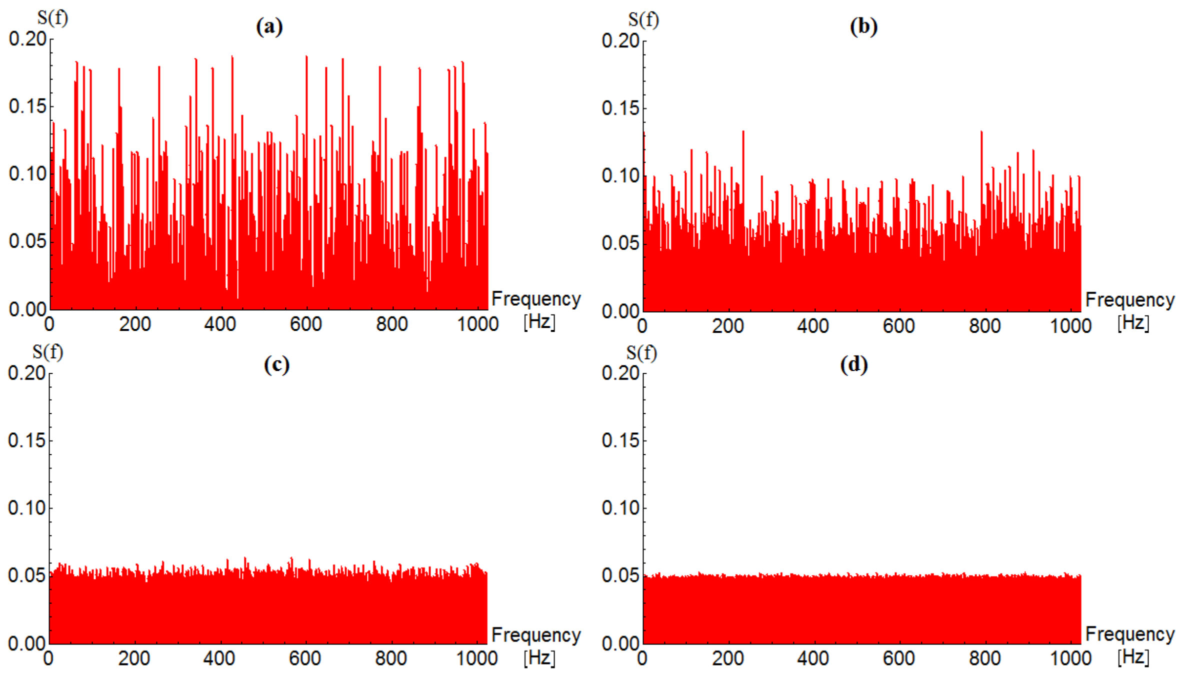


Figure 4.5*: Power spectrum convergence over an ensemble. (a) A single realization with mean .0489 (b) Average over 10 realizations with mean .0477 (c) Average over 100 realizations with mean .0477 (d) Average over 1000 realizations with mean .0479*

Even though the original power spectrum has high variance, its mean value still closely resembles what the ensemble ends up converging to. Both of these values also match the expected value of $\sigma^2\tau_0$ formulated earlier. For this particular signal, $\sigma^2\tau_0$ is approximately .048.

A further investigation can confirm if the correct power spectrum has been estimated. This requires translating Equation (1.10) into Mathematica to produce a covariance sequence. The result of doing so is depicted in Figure 4.6.

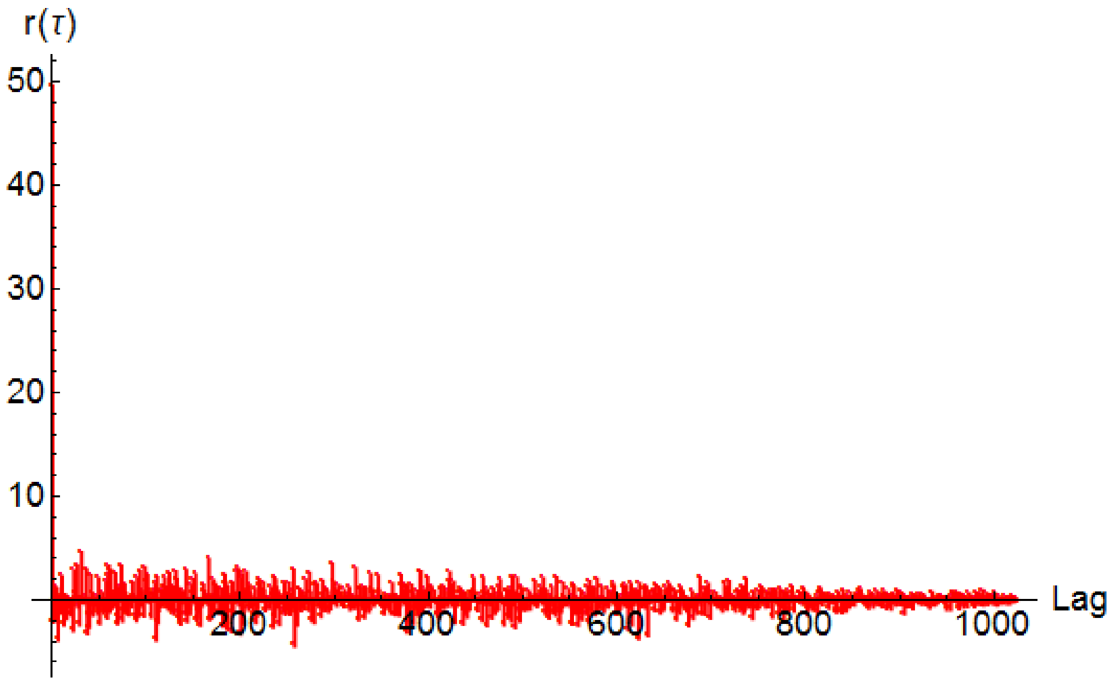


Figure 4.6*: Autocovariance sequence of the generated white noise.*

As predicted for white noise, the autocovariance is a delta spike about zero lag. The Weiner-Khinchin Theorem is utilized (Equation 2.16), and the resulting power spectrum shown in Figure 4.7 is compared to the previous.

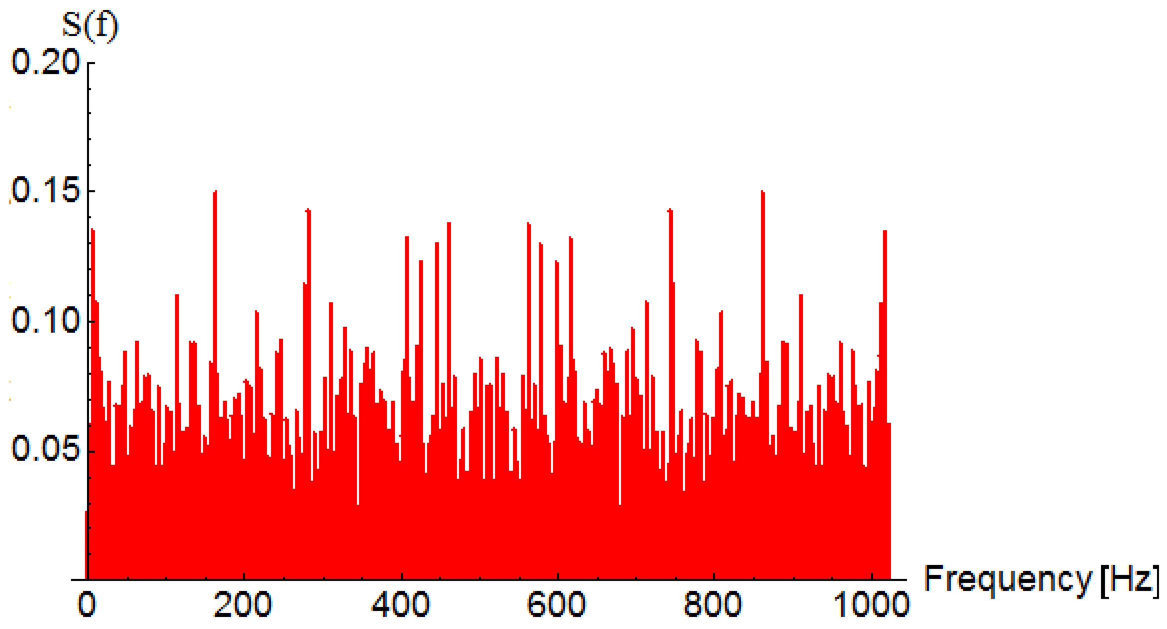


***Figure 4.7**: Power spectrum calculated from the autocovariance sequence.*

This new power spectrum has the same shape as the one in Figure 4.4, but contains less variance. The average value of this spectrum; however, is .048, which is in agreement with the previous results.

4.2 Generating Noise from a Power Spectrum

Because of their distinct power spectra, each noisy signal can be reconstructed when a particular power spectrum is given. The general outline for doing this in Mathematica is as follows. The power spectrum is estimated by squaring the frequency spectrum, and then normalizing it by some value. To get new data back then, a new frequency domain must be reconstructed that is proportional to the square root of the given power spectrum. Take the square root of the power spectrum to yield the amplitude spectrum. In order to be able to transition back into the time domain, some phase information will be required. This was obtained by taking the DFT of another random list of the same length as the power spectrum. Dividing this new transformed data by its conjugate yields random phase data. Next is to convolve the list of random phase data with the computed amplitude spectrum. The result is now a new frequency spectrum constructed from random phases. Following Equation (2.4), the inverse DFT is taken by applying Equation (4.1) with ***a*** and ***b*** set to -1 and 1 respectively. This transforms the data back into the time domain, producing a signal with the same statistical properties as the original.

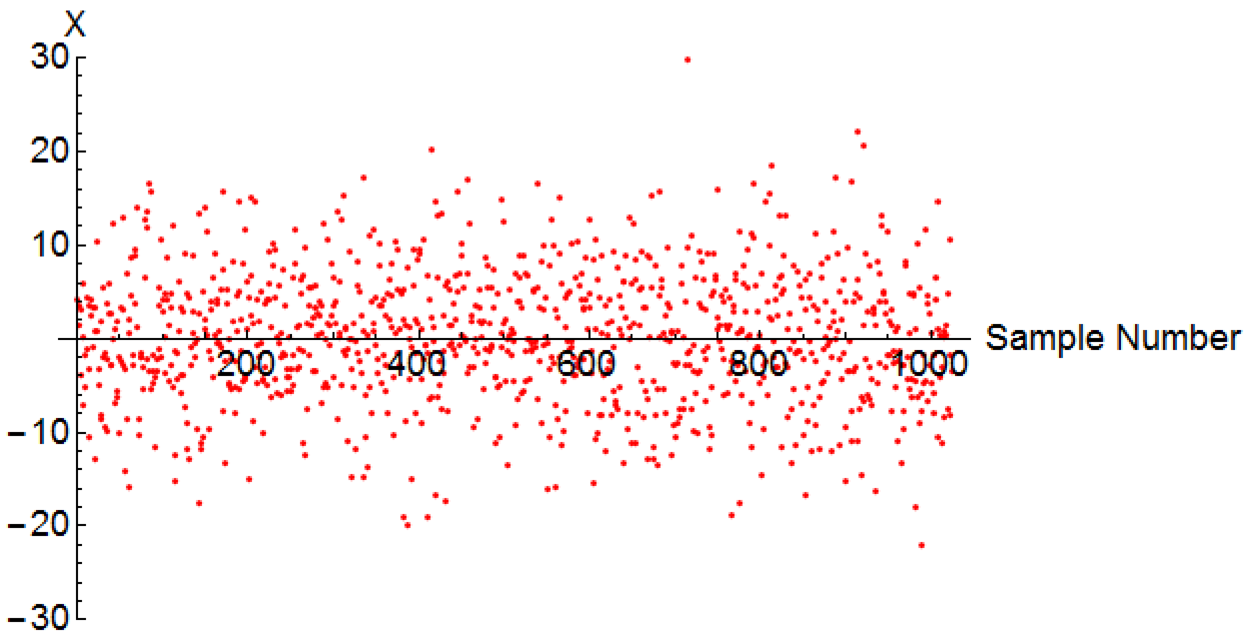


***Figure 4.8**: A new time series reconstructed from the power spectrum.*

The mean of this new list is .333, and it standard deviation is 6.88. These values perfectly match the original signal shown in Figure 4.1. Recalculating the power spectrum of this new signal shows that it is identical to the original power spectrum that produced it.

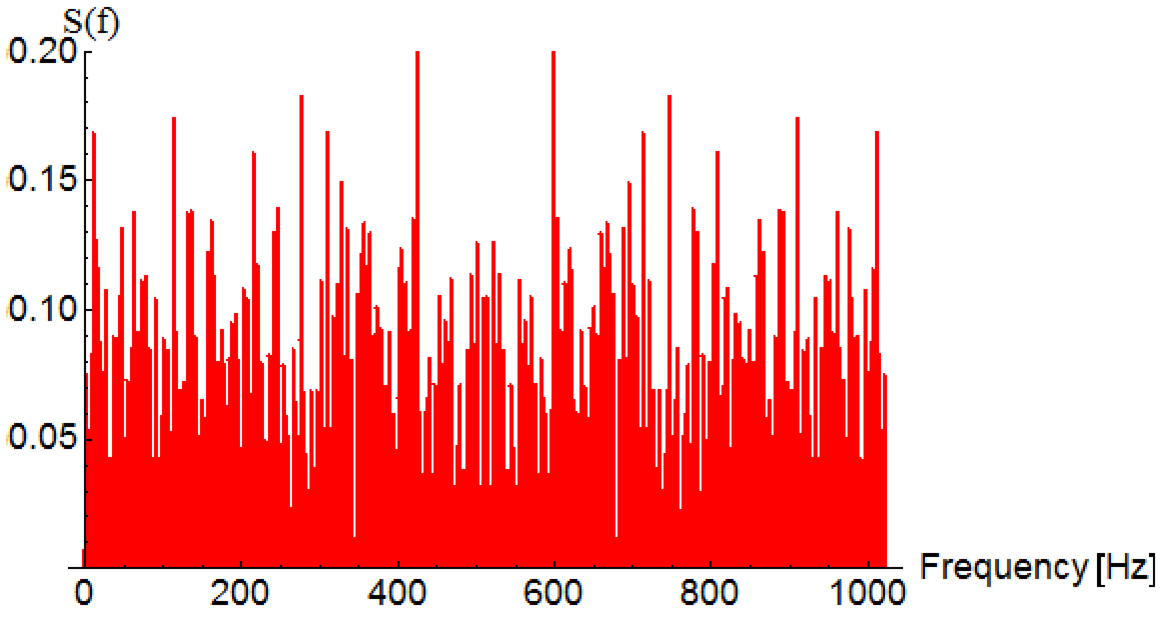


***Figure 4.9**: The power spectrum of the reconstructed signal. It is identical to the original power spectrum.*

4.3 Generating a Time Series for Arbitrary Power Law Noise

In the previous section, a signal was reconstructed from the power spectrum of a single white noise realization. As demonstrated, the spectrum of this single set of data contains fluctuations. Theoretically, white noise is expected to have a flat spectrum. So too do other types of noise spectra have a specific shape as outlined by their respective power law. Recreating a random signal from these spectra requires more steps than in the previous case. This is due to the fact that these power law spectra are only defined between the zero and Nyquist frequencies, whereas the estimated power spectrum of a given signal should be mirrored about the Nyquist frequency. In order to generate an accurate time series that will produce the same spectra, the left and right hand sides of the frequency spectrum will need to be dealt with separately.

The first step remains the same in that the square root of the given power spectrum is calculated. Now, a new list of random numbers that is twice the length of the current power spectrum must be generated. Taking the DFT of this list will result in a frequency spectrum mirrored about the Nyquist frequency. Once again, divide this list by its conjugate to produce random phase data. Only the left half of this list, parts 1 through the Nyquist frequency, will match up with the power spectrum. To obtain individual segments of a list in Mathematica, call the list with the following syntax attached to it: `[List][[First element;;Final element]]`. In this case, the first element needed is in position 1 of the list, and the final element needed occurs at the Nyquist frequency, which is half way through the complete list. Once obtained, the left side can be scaled right away by multiplying it with the square root of the power spectrum. This will form the first half of the new frequency spectrum.

To get the second half, the right hand side of the random phase data must be scaled as well. As mentioned before, the zero and Nyquist frequencies only occur once. So the elements of the list needed to form the right side will begin at the Nyquist frequency + 1, and include all points through the end of the list. Because it is a mirror image of the left, it must first be flipped before scaling it to the power spectrum. This is achieved through using `Reverse[]`. This reversed right hand side can now be scaled by multiplying with the square root of the power spectrum as well. To turn it back into its former mirrored image, `Reverse[]` is applied once more. The now properly scaled left and right hand sides can be brought back together using `Join[]`. What's left is a complete frequency spectrum whose inverse DFT will generate a new random signal. The power spectrum of this newly generated signal can be calculated using the same techniques as before. Doing so shows that the resulting spectrum is identical to the original. Figure 4.10 shows the results of applying this to each of the five power law noise spectra.

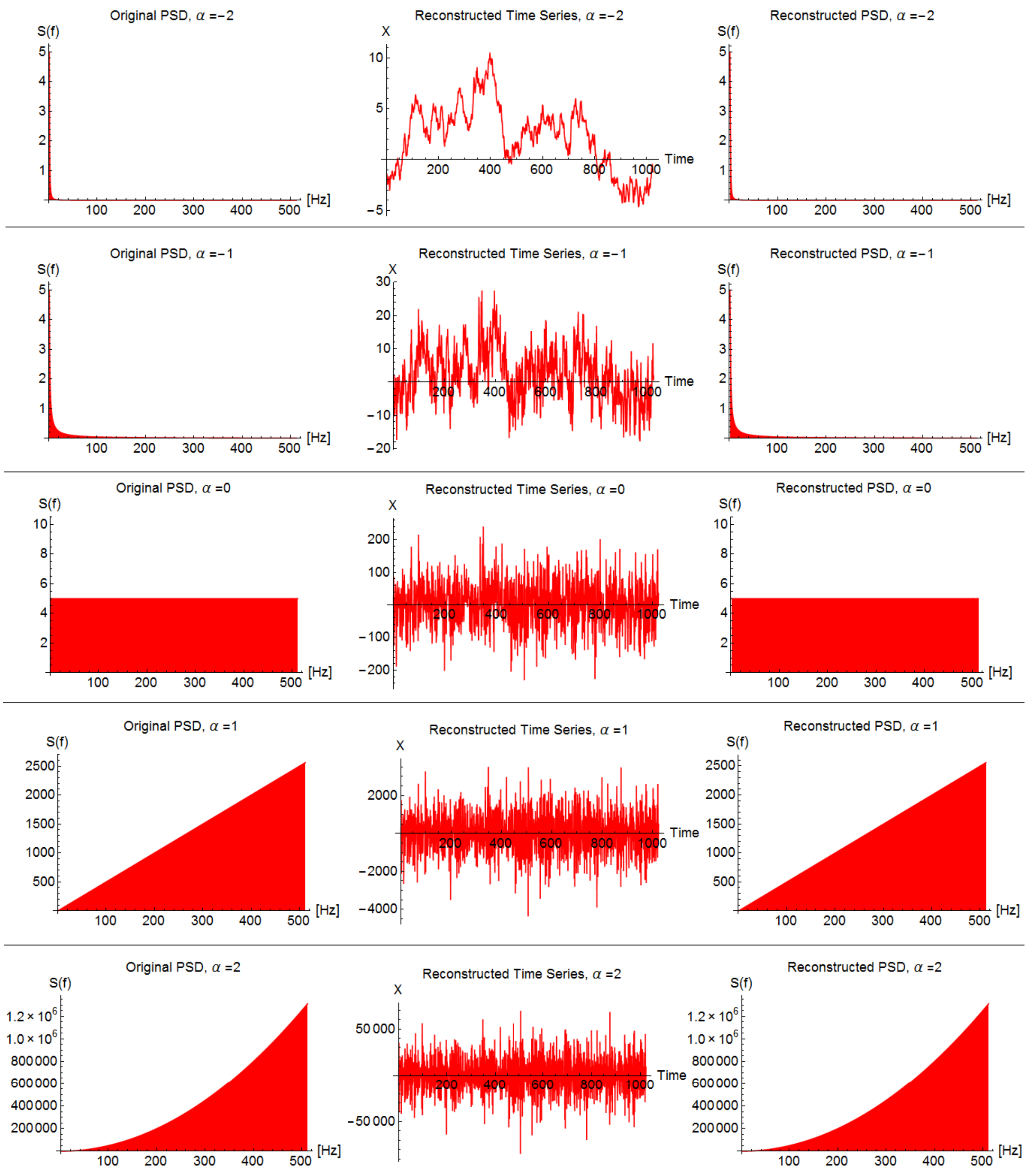


***Figure 4.10**: Generated noise time series and power spectra. The first plot in each row is of a power law noise spectrum. The second and third image are the generated signal and power spectrum respectively. The original power spectra shown here have been produced using the relationship given by Equation (3.4). In each case, h_α has been arbitrarily set to 5.*

4.4 Application

The methods discussed can be directly applied to a clock signal. The bias shown in Figure 1.3 is used as an example. Generating a signal from its power spectrum (shown in Figure 3.6) produces the following random data:

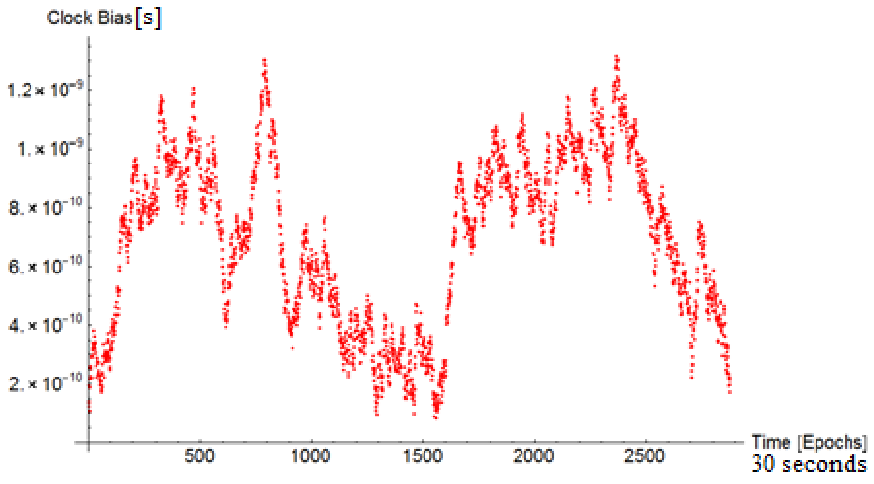


***Figure 4.11**: A randomly generated clock signal.*

It is possible to inject a dark matter signal into the reconstructed phase data. The dark matter signal takes the form of the step function shown in Figure 1.2. It is generated by building a list that is the same length of the phase data, and then setting all values equal to zero, with the exception of a small portion of consecutive terms at any random portion of the list. The amplitude of these terms will reflect the amplitude of the bias caused by the dark matter event. It is not entirely known how much bias is to be expected. For this example, the value is arbitrarily set to 5×10^{-11} *seconds*. An image of the a reconstructed clock signal with an added TDM event is shown in Figure 4.12.

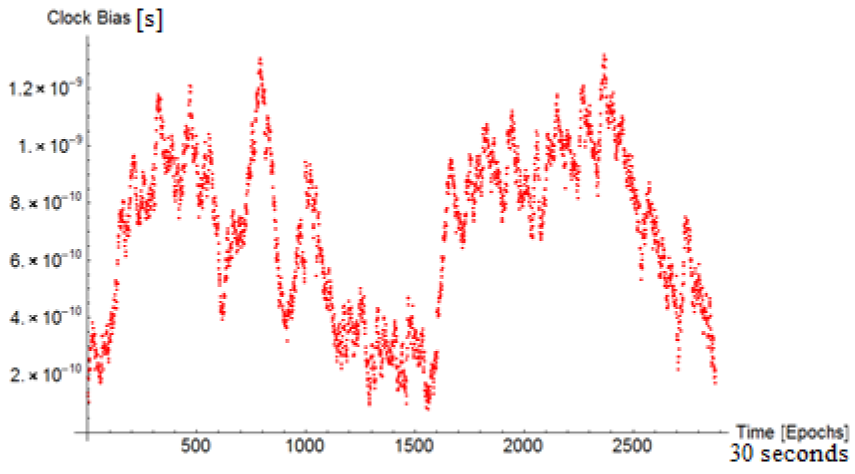


***Figure 4.12**: The generated clock signal from Figure 4.11 with a TDM event added to it. The event has been introduced at the 1000th epoch.*

Though not obvious at first, the dark matter signal for this example begins at the 1000th epoch. Double differencing of the data reduces the clock data into Gaussian white noise. This is plotted in Figure 4.13.

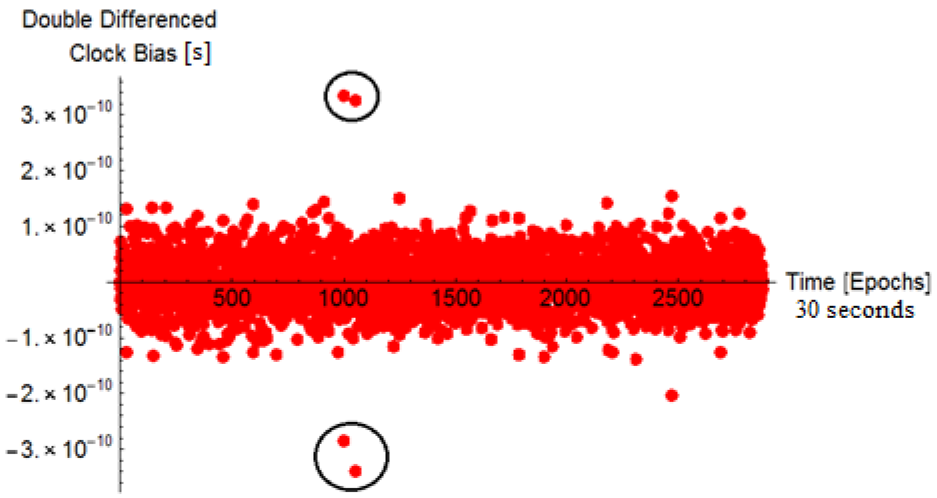


Figure 4.13*: Double differenced clock bias. The TDM event has been circled in black.*

The standard deviation of the white noise produced is 4.56×10^{-11}. Most of the data lies within three standard deviations, as predicted for a Gaussian distribution. The event makes its appearance as a jump that is nearly seven standard deviations from the mean. For a large enough jumps, then, the event can be clearly seen. Despite this, such a jump must be seen in other satellite clocks at specific times in order to adequately confirm an event.

A question that might arise is whether the signature detected in the double differenced data was actually introduced by the injected dark matter signal, or if such a signal was already present. Reproducing clock bias that contains a dark matter signal effectively creates a 'blank slate' to work with. This is illustrated by applying the methods outlined in Section 4.2 to the signal shown in Figure 4.12. There is a known event in this signal because it was purposefully placed there. Generating a new time series from the power spectrum of this data produces the following random signal:

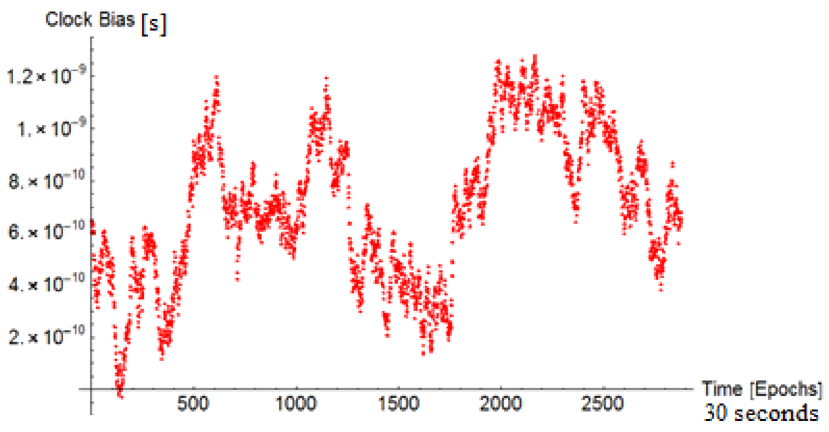


Figure 4.14*: A random signal generated from bias that contained a TDM event.*

This new signal can be double differenced as before to see if the dark matter signal can still be detected. The results are shown in Figure 4.15.

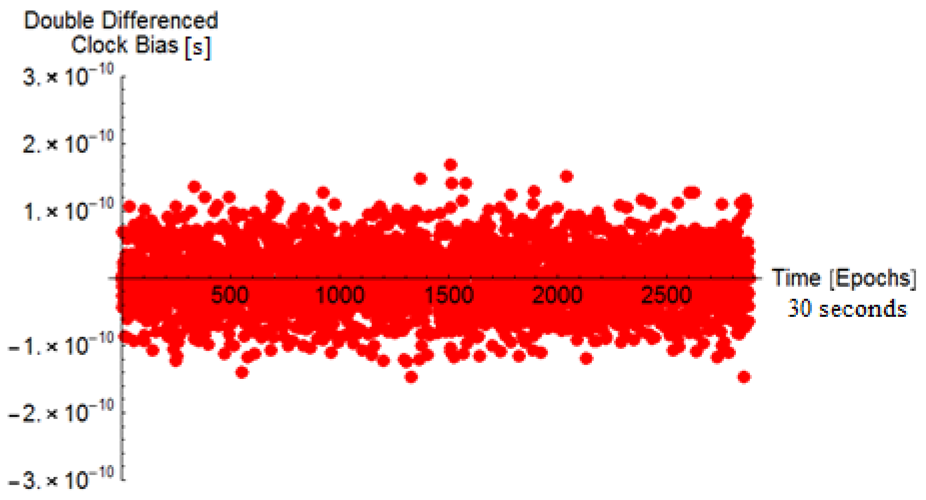


***Figure 4.15**: Double differenced clock bias of a signal reconstructed from data containing a TDM event.*

Once again, the standard deviation of the white noise produced is 4.56×10^{-11}. Again, most of the data lies within three standard deviations from the mean. The jump seen before at the 1000th epoch is no longer present. This demonstrates that producing a new signal with the given methods eliminates events of a large enough magnitude.

Chapter 5

Conclusion

The numerical results of this research are promising. Producing a power spectrum through the periodogram leads to high variance in the estimate. However, an average over an ensemble of noise realizations causes the power spectrum to converge to its actual value. Calculating the power spectrum from the autocovariance sequence leads to less variance, but it too converges to the same value. This sequence does not exist for nonstationary stochastic processes such as random walk. This can be remedied by differencing these noise types to produce white noise. In the case of white noise, it has been demonstrated that a power spectrum produced from a random signal can also be used to generate a new time series. This newly generated data exhibits the same statistical properties as the original signal. This is a desirable outcome, as it means these techniques can be used to generate atomic clock bias with the appropriate variance.

These methods will allow signals to be generated that emulate atomic clock data. A library of these signals will be developed for each type of clock on board GPS satellites in orbit. The proposed dark matter signal can then be introduced into these lists of random data. If the signal to noise ratio is strong enough, then the signal can be recovered. These will effectively form a simulation of what to look for during a topological dark matter event. When considering dark matter candidates in actual clock data, the simulation can be used to cross compare results.

There are two key benefits to being able to generate random clock data. The first has to do with logistics. It takes time to collect, import, and process clock data. Now, it can simply be generated in a fraction of the time. In addition to this, the methods discussed produce new clock data sets that are free from existing dark matter signals. This provides a ‘clean slate’ to work with when injecting dark matter signals into the data.

Bibliography

1. Erickson, K. (2015, June 5). Dark Energy, Dark Matter - NASA Science. Retrieved April 4, 2016, from `http://science.nasa.gov/astrophysics/focus-areas/what-is-dark-energy/`

2. Derevianko, A., & Pospelov, M. (2014). Hunting for topological dark matter with atomic clocks. *Nature Physics*, *10*(12), 933–936. `http://doi.org/10.1038/NPHYS3137`

3. Maoz, D. (2007). *Astrophysics in a Nutshell*. 41 William Street, Princeton, NJ 08540: Princeton University Press.

4. Space Segment. Retrieved April 21, 2016, from `http://www.gps.gov/systems/gps/space/#content`

5. Dwyer, D. (2000, April). How Atomic Clocks Work. Retrieved April 2, 2016, from `http://science.howstuffworks.com/atomic-clock.htm`

6. Griffiths, D. (2005). *Introduction to Quantum Mechanics* (2nd ed.). Upper Saddle River, NJ 07458: Pearson Education.

7. Hecht, E. (2002). *Optics* (4th ed.). San Francisco, CA: Pearson Education.

8. Base unit definitions: Second. Retrieved April 2, 2016, from `http://physics.nist.gov/cuu/Units/second.html`

9. Walpole, R., Myers, R., Myers, S., & Ye, K. (2007). *Probability and Statistics for Engineers and Scientists* (8th ed.). Upper Saddle River, NJ 07458: Pearson Education. Retrieved from `https://drive.google.com/file/d/0B6ZNdXkCQbsuZDJkNDQ1YTItMjQxMy00ZTVhLTk0YTMtYmJjNWYwYTRjYzNj/view?usp=drive_web&ddrp=1&usp=embed_facebook`

10. Riley, W. J. (2008). Handbook of Frequency Stability Analysis. Retrieved from `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication1065.pdf`

11. Smith, S. (1999). *The Scientist and Engineer's Guide to Digital Signal Processing* (2nd ed.). San Diego, California: California Technical Publishing.

12. Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in Fortran* (2nd ed.). Cambridge University Press.

13. Gregory, P. (2005). *Bayesian Logical Data Analysis for the Physical Sciences*. Cambridge, New York: Cambridge University Press.

14. Noise and Noise Rejection. Retrieved from `https://engineering.purdue.edu/ME365/Textbook/chapter11.pdf`

15. Barnes, J. et al. (1971). Characterization of Frequency Stability. *IEEE Transactions on Instrumentation and Measurement*, *20*(2). Retrieved from `http://tf.nist.gov/general/tn1337/Tn146.PDF`

16. The Science of Timekeeping. (1997). Retrieved from `http://itsabouttimebook.com/wp-content/uploads/2016/02/science-of-keeping-time.pdf`

Appendices

These source codes are available for the GPS.DM collaboration in the shared directory:

https://www.dropbox.com/home/GPS.DM/Students/AlexRollings

Appendix A

Mathematica Code for Gaussian White Noise

Gaussian White Noise Sampled at Discrete Time Intervals

Define Parameters

Choose Total duration of time the signal is sampled for (seconds).

```
T = 1;
```

Choose Sampling Frequency: Samples per second (Hz). Must be a power of 2 to compute.

```
sf = 2^10;
```

Sampling Period: Time interval between samples (seconds).

```
τ0 = 1 / sf;
```

Total number of data points generated.

```
npts = T * sf;
```

Nyquist Frequency (Hz).

```
nyq = npts / 2;
```

Choose the Mean value of the data.

```
mean = 0;
```

Choose the Standard Deviation of the data.

```
σ = 7;
```

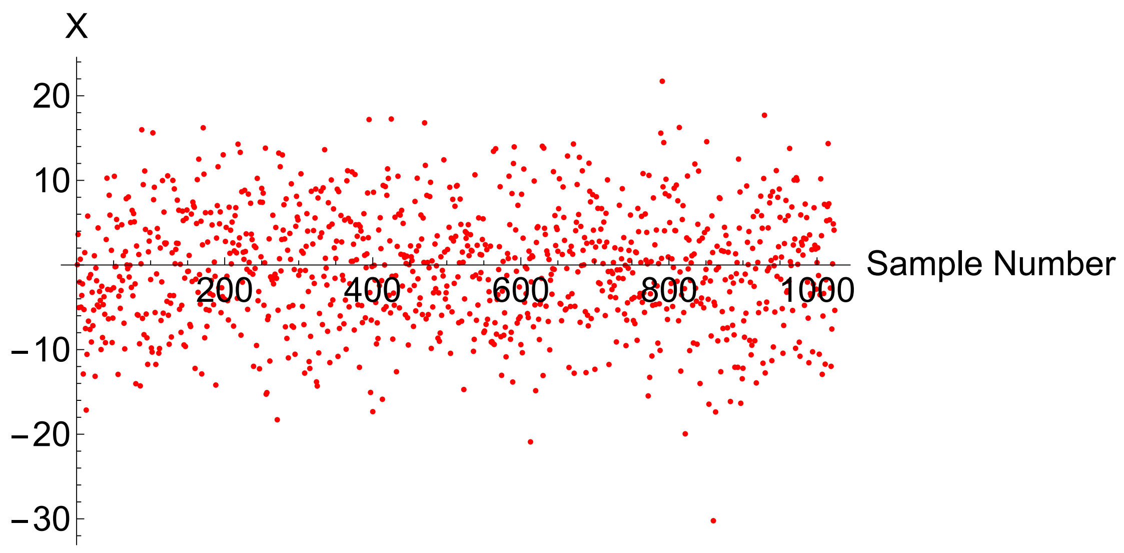
Generate Gaussian White Noise in Time Domain

Generate Normally Distributed Random Data

```
xlst = RandomReal[NormalDistribution[mean, σ], npts];
```

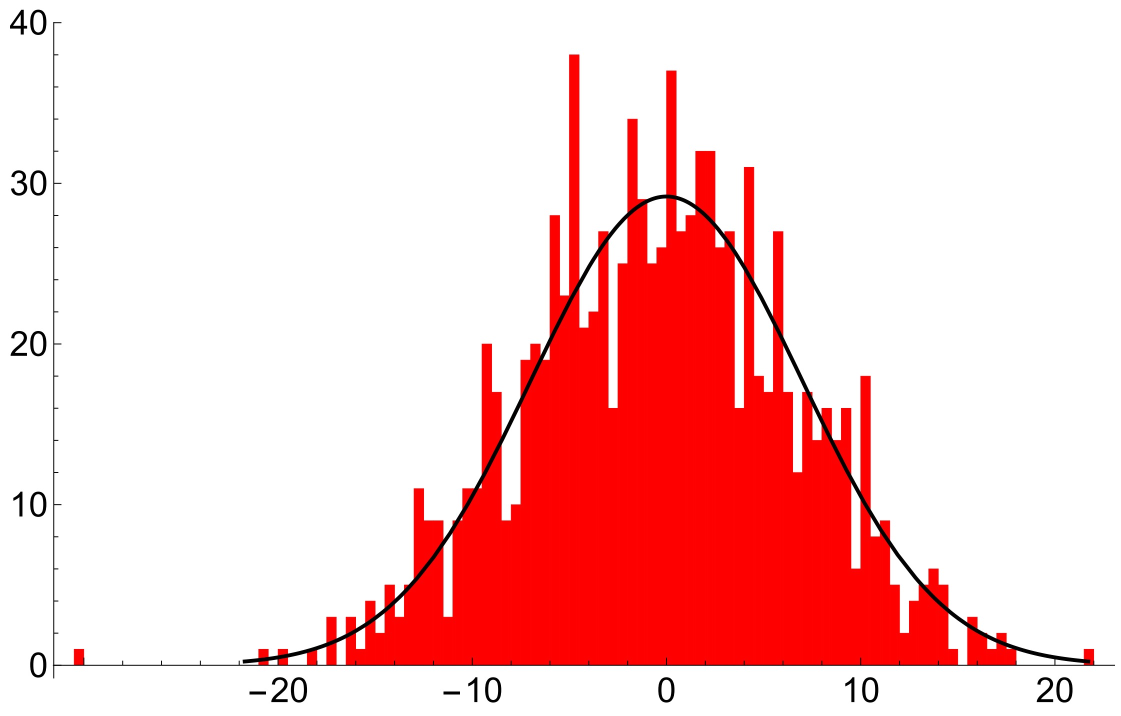
Plot as time series

```
ListPlot[xlst, PlotRange → All, AxesLabel → {"Sample Number", "X"},
 LabelStyle → Directive[Black, 18], ImageSize → Large, PlotStyle → Red,
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



Show histogram of data

```
hist = Histogram[xlst, 100, ChartStyle → Red,
   AxesStyle → Black, TicksStyle → Directive[Black, 18]];
pdf = Plot[(npts / 2) * PDF[NormalDistribution[mean, σ], x],
   {x, -Max[xlst], Max[xlst]}, PlotStyle → Directive[Thick, Black]];
Show[{hist, pdf}, ImageSize → Large]
```



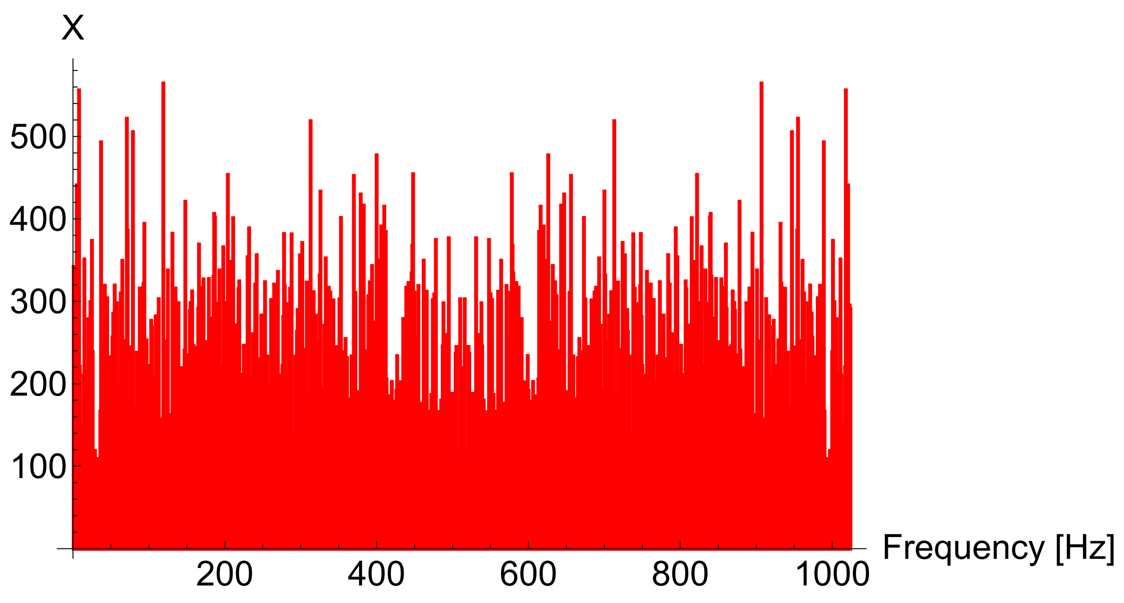
Analyze Frequency Spectrum

Take Discrete Fourier Transform of data

```
dft = Fourier[xlst, FourierParameters → {1, -1}];
flst = Abs[dft];
```

Plot the magnitude of the frequency spectrum

```
ListPlot[flst, PlotRange → All,
 AxesLabel → {"Frequency [Hz]", "X"}, LabelStyle → Directive[Black, 18],
 Filling → Axis, FillingStyle → Directive[Thick, Red],
 ImageSize → Large, PlotMarkers → Style["●", 3, Red],
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



Proof of Parseval's Theorem

```
energy1 = Sum[xlst[[n]]^2, {n, 1, npts}];
energy2 = (1 / npts) * Sum[flst[[n]]^2, {n, 1, npts}];
StringForm["Total energy in time domain = ``", energy1]
StringForm["Total energy in frequency domain = ``", energy2]
```

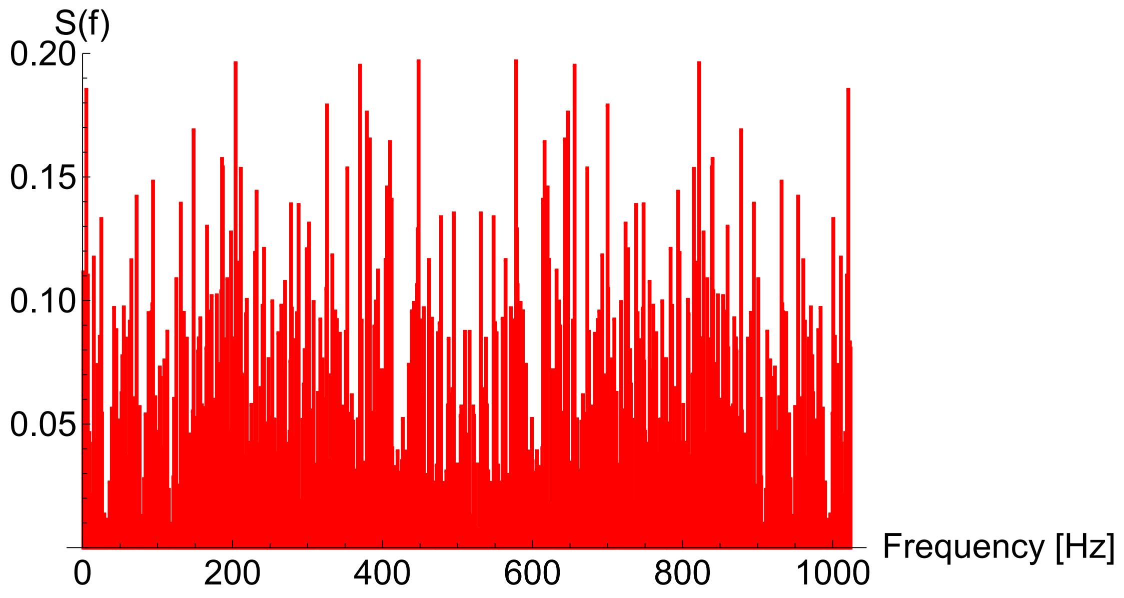
```
Total energy in time domain = 48604.77548521312`

Total energy in frequency domain = 48604.77548521318`
```

Estimate Power Spectral Density (PSD)

```
psd1 = (τ0 / npts) * flst^2;
```

Plot Power Spectrum

```
ListPlot[psd1, PlotRange → .2,
 AxesLabel → {"Frequency [Hz]", "S(f)"}, LabelStyle → Directive[Black, 18],
 Filling → Axis, FillingStyle → Directive[Thick, Red],
 ImageSize → Large, PlotMarkers → Style["●", 3, Red],
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



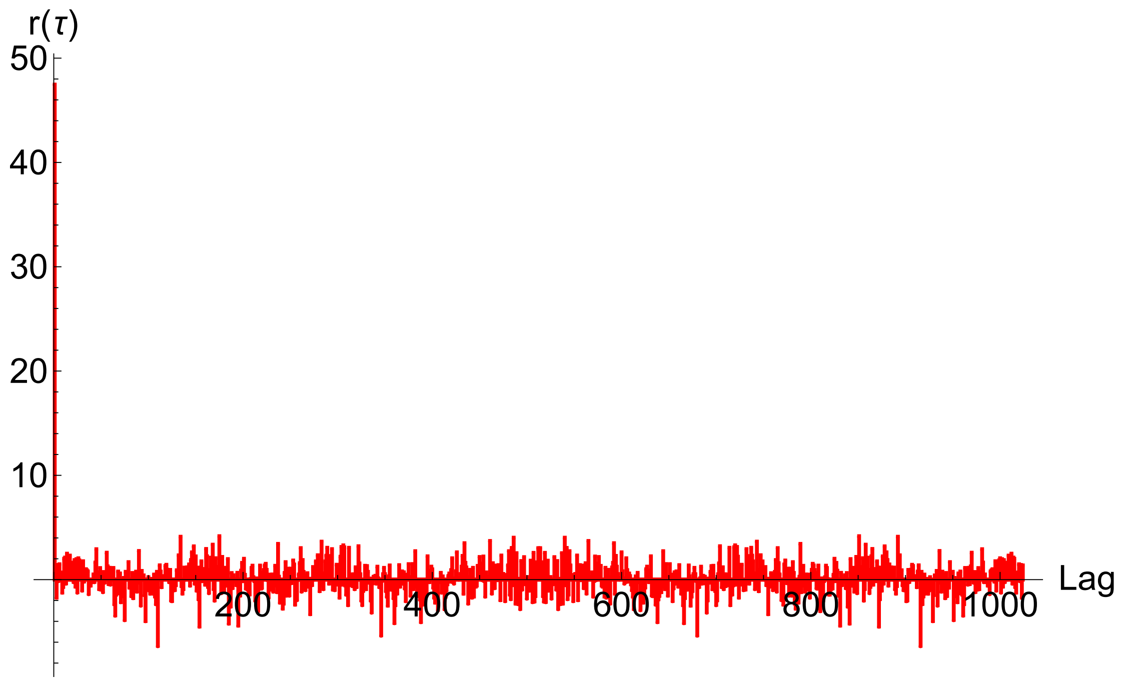
Wiener–Khinchin Theorem

Generate auto-correlation function (ACF) by taking the inverse DFT of the PSD (the real part of the transform is taken because *Mathematica* introduces minor error in the form of negligibly small imaginary terms)

```
acf1 = (1 / T) * Re[Fourier[psd1, FourierParameters → {1, 1}]];
```

Plot the ACF

```
ListPlot[acf1, PlotRange → All,
 AxesLabel → {"Lag", "r(τ)"}, LabelStyle → Directive[Black, 18],
 Filling → Axis, FillingStyle → Directive[Thick, Red],
 ImageSize → Large, PlotMarkers → Style["●", 3, Red],
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```

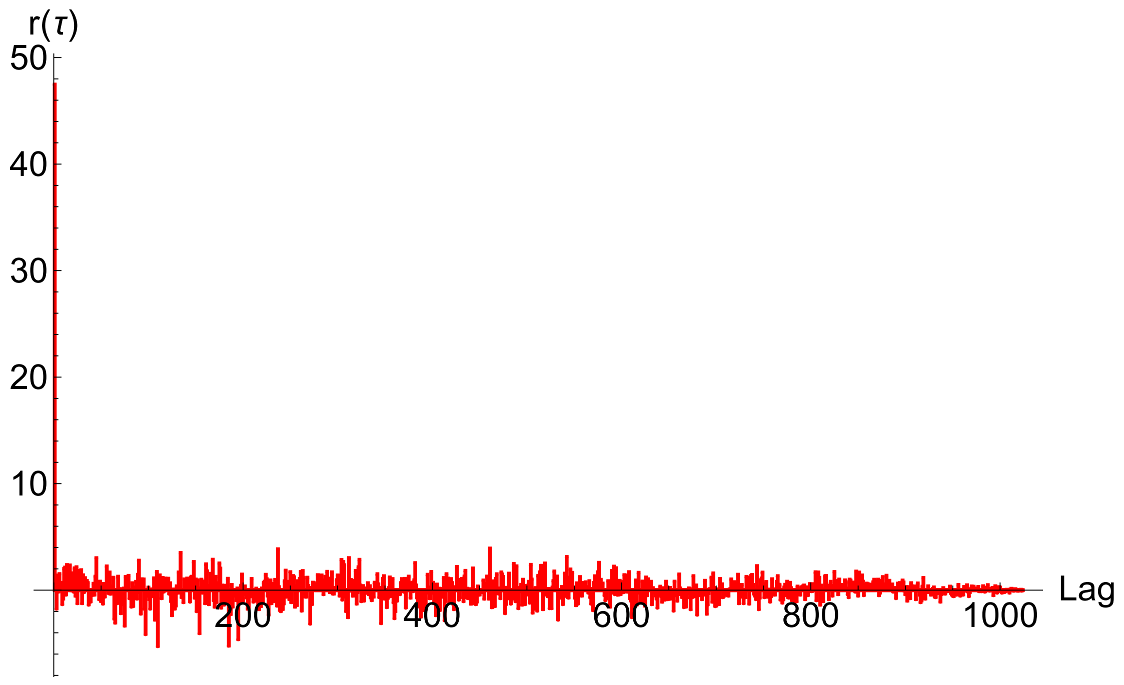


Check by computing the ACF directly

```
acf2 =
  Table[(1 / npts) * Sum[(xlst[[n]] - mean) * (xlst[[n + lag]] - mean), {n, 1, npts - lag}],
   {lag, 0, npts - 1}];
```

Plot the new ACF

```
ListPlot[acf2, PlotRange → All,
 AxesLabel → {"Lag", "r(τ)"}, LabelStyle → Directive[Black, 18],
 Filling → Axis, FillingStyle → Directive[Thick, Red],
 ImageSize → Large, PlotMarkers → Style["●", 3, Red],
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```

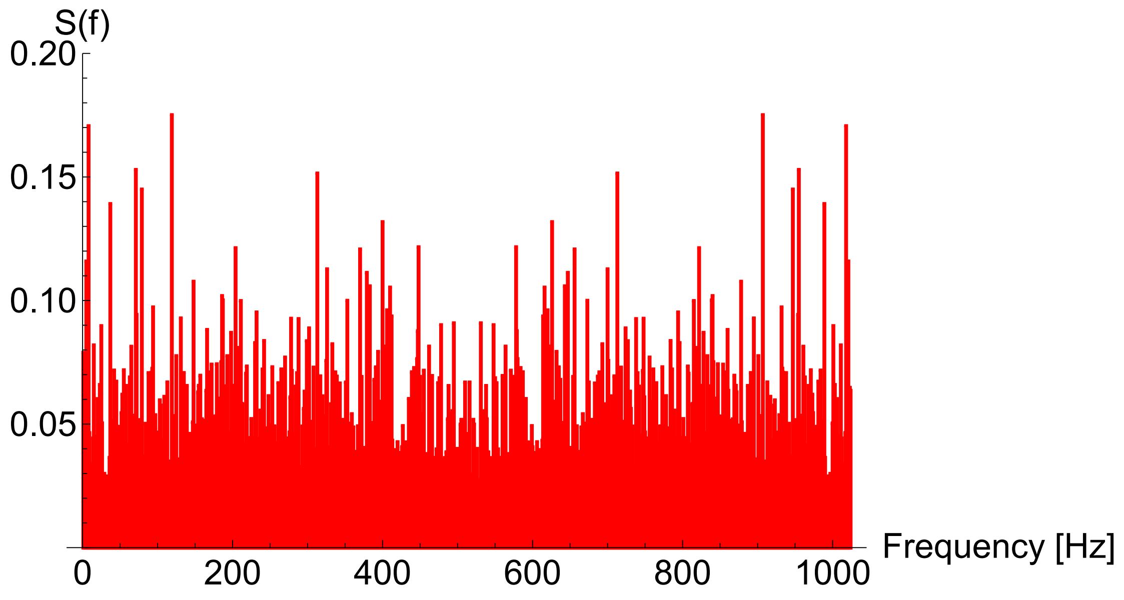


Estimate PSD from the DFT of the ACF

```
psd2 = Re[Fourier[acf2, FourierParameters → {-1, -1}]];
```

Plot the new PSD

```
ListPlot[psd2, PlotRange → .2,
 AxesLabel → {"Frequency [Hz]", "S(f)"}, LabelStyle → Directive[Black, 18],
 Filling → Axis, FillingStyle → Directive[Thick, Red],
 ImageSize → Large, PlotMarkers → Style["●", 3, Red],
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



Check that each spectra converges to the same value

```
StringForm["Mean of first PSD = ``", Mean[psd1]]
StringForm["Mean of second PSD = ``", Mean[psd2]]
StringForm["σ²τ₀ = ``", σ² τ₀ // N]
```

```
Mean of first PSD = 0.046353126034939934`

Mean of second PSD = 0.04635312603493989`

σ²τ₀ = 0.0478515625`
```

Generate New Signal From the Existing Power Spectrum

Produce a random time series of the same length as number of frequency bins (the data can have any statistical properties)

```
rdata = RandomReal[{-1, 1}, Length[psd1]];
```

Take DFT of the new random data

```
fdata = Fourier[rdata, FourierParameters → {1, -1}];
```

Divide the transformed data by its conjugate to produce random phase data

```
conj = Conjugate[fdata];
phase = fdata / conj;
```

Take square root of PSD to get the amplitude spectrum

```
amplitude = Sqrt[psd1];
```

Multiply the amplitude spectrum by the random phase data to produce a new frequency spectrum

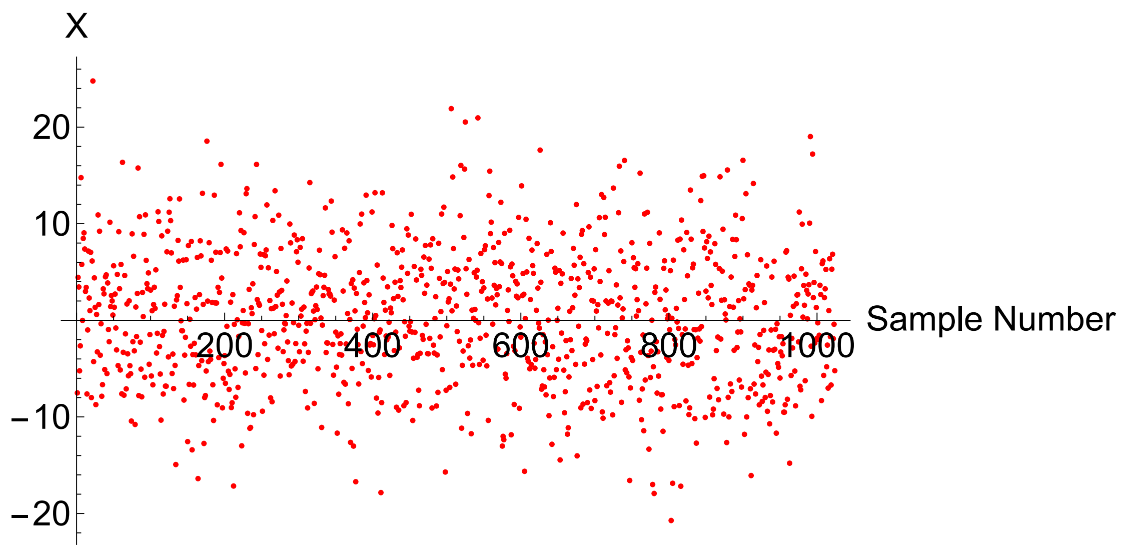
```
newfreq = amplitude * phase;
```

Take inverse DFT to reproduce a new time series (take real part to eliminate imaginary terms introduced through roundoff errors)

```
newdata = Re[Fourier[newfreq, FourierParameters → {1, 1}]];
```

Plot new data

```
ListPlot[newdata, PlotRange → All, AxesLabel → {"Sample Number", "X"},
 LabelStyle → Directive[Black, 18], ImageSize → Large, PlotStyle → Red,
 AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



Check that the statistical properties of the reconstructed data match the original data

```
StringForm["Mean of original data = ``", Mean[xlst]]
StringForm["Mean of reconstructed data = ``", Mean[newdata]]
StringForm["σ of original data = ``", StandardDeviation[xlst]]
StringForm["σ of reconstructed data = ``", StandardDeviation[newdata]]
```

```
Mean of original data = -0.333742

Mean of reconstructed data = 0.333742118537928`

σ of original data = 6.884802598152638`

σ of reconstructed data = 6.884802598152637`
```

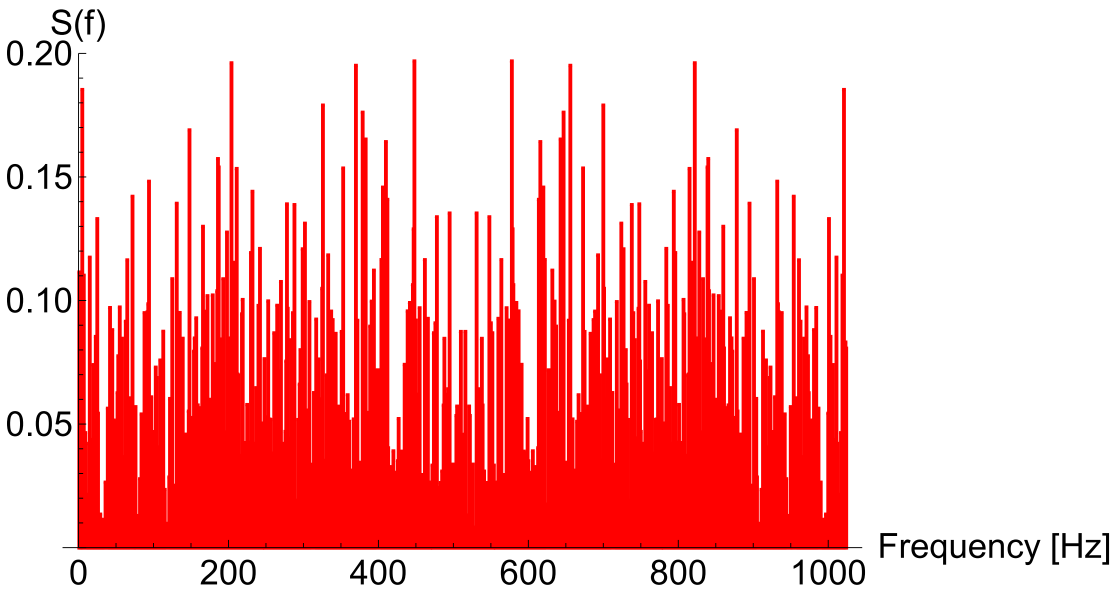
Check that new PSD equals the original PSD

```
newpsd = (τ0 / npts) * Abs[Fourier[newdata, FourierParameters → {1, -1}]]^2;
StringForm["Mean of original PSD = ``", Mean[psd1]]
StringForm["Mean of new PSD = ``", Mean[newpsd]]
```

```
Mean of original PSD = 0.046353126034939934`

Mean of new PSD = 0.04635312603493991`
```

Plot the new PSD

```
p1 = ListPlot[newpsd, PlotRange → .2,
  AxesLabel → {"Frequency [Hz]", "S(f)"}, LabelStyle → Directive[Black, 18],
  Filling → Axis, FillingStyle → Directive[Thick, Red],
  ImageSize → Large, PlotMarkers → Style["●", 3, Red],
  AxesStyle → Black, TicksStyle → Directive[Black, 18]]
```



Appendix B

Mathematica Code for Reconstructing Power Law Noise

Reconstruct a Time Series from a Given Power Law PSD

Generate the Power Spectrum

```
In[93]:= Clear["Global`*"]
```

Create a list of distinct frequencies that will go up to the Nyquist frequency (in Hz).

```
In[94]:= nyq = 512;
         freq = Range[1, nyq];
```

Raise the frequencies to a power, α, in accordance with the noise type.

```
In[96]:= freqExp = freq^α;
```

Choose an initial amplitude h_0 for the power spectrum.

```
In[97]:= h0 = 5;
```

Generate the power spectrum.

```
In[98]:= psd1 = h0 * freqExp;
```

Take square root of power spectrum to get the amplitude spectrum.

```
In[99]:= sqrtpsd = Sqrt[psd1];
```

Create Random Frequency Data

Generate a list of random data in the time domain that is twice as long as the power spectrum.

```
In[100]:= xlst = RandomReal[{-1, 1}, 2 * nyq];
```

Take the DFT of the random data.

```
In[101]:= flst = Fourier[xlst, FourierParameters → {1, -1}];
```

Divide the frequency spectrum by its conjugate to get random phase data.

```
In[102]:= conj = Conjugate[flst];
          phase = flst / conj;
```

The frequency spectrum is symmetric about the Nyquist frequency. Divide the list into the two halves. Note: the first frequency component and the Nyquist frequency only show up once.

```
In[104]:= leftSide = phase[[1 ;; nyq]];
          rightSide = phase[[nyq + 1 ;;]];
```

Generate the Time Series

The left hand side should be proportional to the amplitude spectrum. The right hand side will be proportional to the mirror of the amplitude spectrum.

```
In[106]:= leftFreq = leftSide * sqrtpsd;
          rightFreq = rightSide * Reverse[sqrtpsd];
```

Re-join both halves to get the new frequency spectrum.

```
In[108]:= totalFreq = Join[leftFreq, rightFreq];
```

Inverse DFT to get the new time series (take real part to eliminate imaginary terms introduced through roundoff error).

```
In[109]:= newData = Re[Fourier[totalFreq, FourierParameters → {1, 1}]];
```

Fourier::fftl : Argument $\{2.23607 + 0.\,i, (1.59838 + 1.56371\,i)\sqrt{2^{\alpha}}, (1.17277 + 1.90384\,i)\sqrt{3^{\alpha}}, (0.5274 + 2.17298\,i)\sqrt{4^{\alpha}}, (-0.295376 + 0.955381\,i)\sqrt{5^{1+\alpha}}, (1.91712 - 1.15093\,i)\sqrt{6^{\alpha}}, (1.90532 + 1.17036\,i)\sqrt{7^{\alpha}}, \ll 37 \gg, (-0.612171 - 0.790726\,i)\sqrt{5^{1+\alpha}\,9^{\alpha}}, (-1.99495 + 1.01004\,i)\sqrt{46^{\alpha}}, (0.608733 + 2.15161\,i)\sqrt{47^{\alpha}}, (2.23606 + 0.00448652\,i)\sqrt{48^{\alpha}}, (2.16574 - 0.556392\,i)\sqrt{49^{\alpha}}, (-0.545005 - 0.838433\,i)\sqrt{2^{\alpha}\,5^{1+2\,\alpha}}, \ll 974 \gg\}$

is not a non−empty list or rectangular array of numeric quantities. ≫

Check that the power spectrum of the new data is equal to the original power spectrum.

```
In[110]:= psd2 = Abs[Fourier[newData, FourierParameters → {-1, -1}]]^2;
```

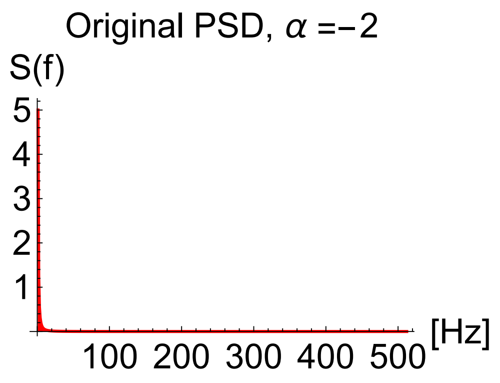
Fourier::fftl : Argument $\mathrm{Re}[\mathrm{Fourier}[\{2.23607 + 0.\,i, (1.59838 + 1.56371\,i)\sqrt{2^{\alpha}}, (1.17277 + 1.90384\,i)\sqrt{3^{\alpha}}, (0.5274 + 2.17298\,i)\sqrt{4^{\alpha}}, (-0.295376 + 0.955381\,i)\sqrt{5^{1+\alpha}}, (1.91712 - 1.15093\,i)\sqrt{6^{\alpha}}, (1.90532 + 1.17036\,i)\sqrt{7^{\alpha}}, \ll 37 \gg, (-0.612171 - 0.790726\,i)\sqrt{5^{\mathrm{Plus}[\ll 2 \gg]}\,9^{\alpha}}, (-1.99495 + 1.01004\,i)\sqrt{46^{\alpha}}, (0.608733 + 2.15161\,i)\sqrt{47^{\alpha}}, (2.23606 + 0.00448652\,i)\sqrt{48^{\alpha}}, (2.16574 - 0.556392\,i)\sqrt{49^{\alpha}}, (-0.545005 - 0.838433\,i)\sqrt{2^{\alpha}\,5^{\mathrm{Plus}[\ll 2 \gg]}}, \ll 974 \gg\}, \mathrm{FourierParameters} \to \ll 1 \gg]]$

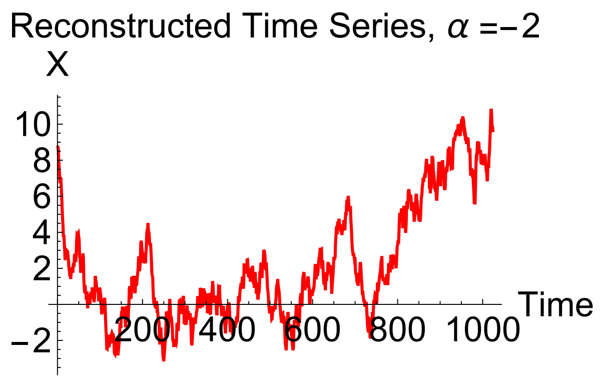
is not a non−empty list or rectangular
array
of
numeric
quantities.
≫

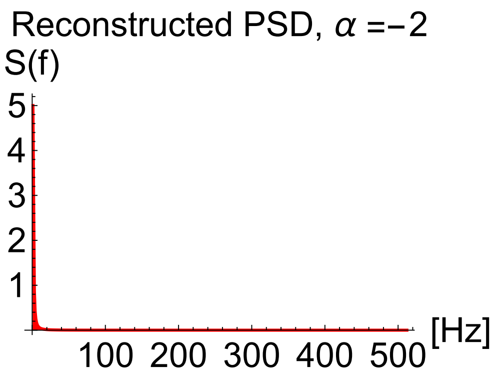
Plot the Results

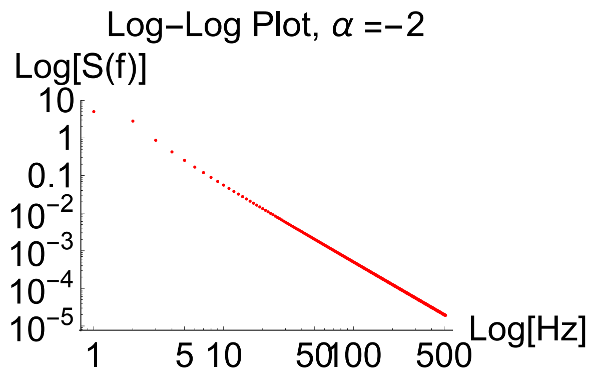
Run the code for the frequency powers from -2 to 2.

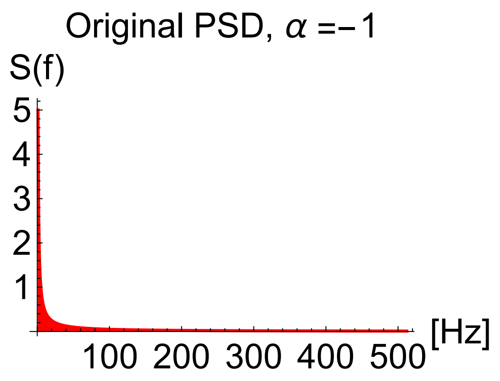
```
In[111]:= For[α = -2, α ≤ 2, α++,
  p1 = ListLinePlot[psd1, PlotRange → All,
    AxesLabel → {"[Hz]", "S(f)"}, LabelStyle → Directive[Black, 18],
    PlotStyle → Red, AxesStyle → Black, TicksStyle → Directive[Black, 18],
    PlotLabel → Style[StringForm["Original PSD, α =``", α], 18],
    Filling → Axis, FillingStyle → Red];
  p2 = ListLinePlot[newData, PlotRange → All, AxesLabel → {"Time", "X"},
    LabelStyle → Directive[Black, 18], PlotStyle → Red,
    AxesStyle → Black, TicksStyle → Directive[Black, 18],
    PlotLabel → Style[StringForm["Reconstructed Time Series, α =``", α], 18]];
  p3 = ListLinePlot[psd2[[1 ;; nyq]], PlotRange → All,
    AxesLabel → {"[Hz]", "S(f)"}, LabelStyle → Directive[Black, 18],
    PlotStyle → Red, AxesStyle → Black, TicksStyle → Directive[Black, 18],
    PlotLabel → Style[StringForm["Reconstructed PSD, α =``", α], 18],
    Filling → Axis, FillingStyle → Red];
  p4 = ListLogLogPlot[psd2[[1 ;; nyq]], PlotRange → All,
    AxesLabel → {"Log[Hz]", "Log[S(f)]"}, LabelStyle → Directive[Black, 18],
    PlotStyle → Red, AxesStyle → Black, TicksStyle → Directive[Black, 18],
    PlotLabel → Style[StringForm["Log-Log Plot, α =``", α], 18]];
  Print[GraphicsRow[{p1, p2}, ImageSize → Full],
   GraphicsRow[{p3, p4}, ImageSize → Full]]]
```

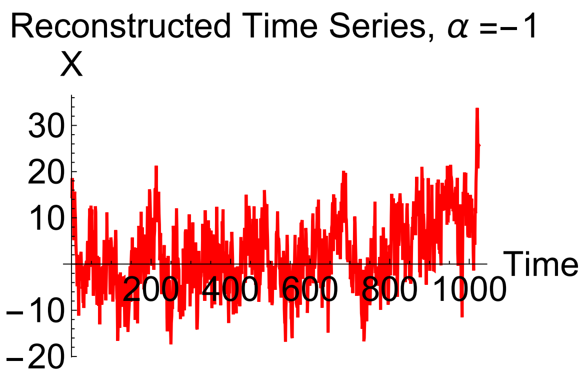

Original PSD, α =−2
S(f)
5
4
3
2
1
100 200 300 400 500
[Hz]

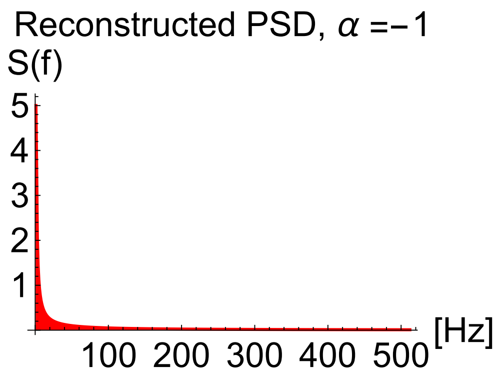

Reconstructed Time Series, α =−2
X
10
8
6
4
2
−2
200 400 600 800 1000
Time

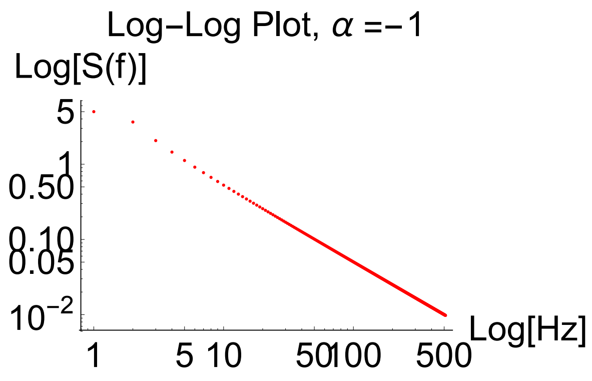

Reconstructed PSD, α =−2
S(f)
5
4
3
2
1
100 200 300 400 500
[Hz]

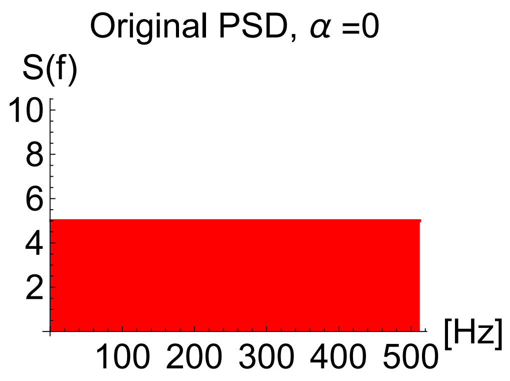

Log−Log Plot, α =−2
Log[S(f)]
10
1
0.1
10^−2
10^−3
10^−4
10^−5
1 5 10 50 100 500
Log[Hz]

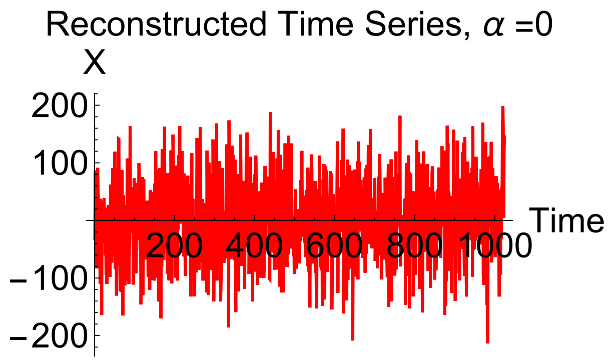

Original PSD, α =−1
S(f)
5
4
3
2
1
100 200 300 400 500
[Hz]

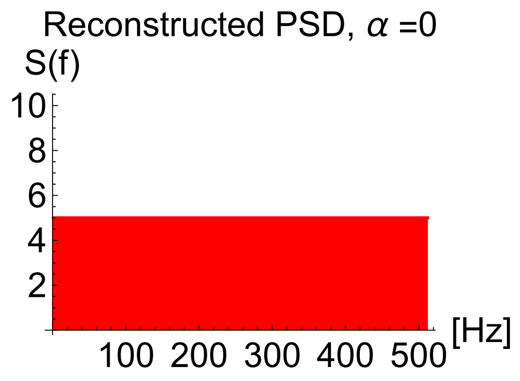

Reconstructed Time Series, α =−1
X
30
20
10
−10
−20
200 400 600 800 1000
Time

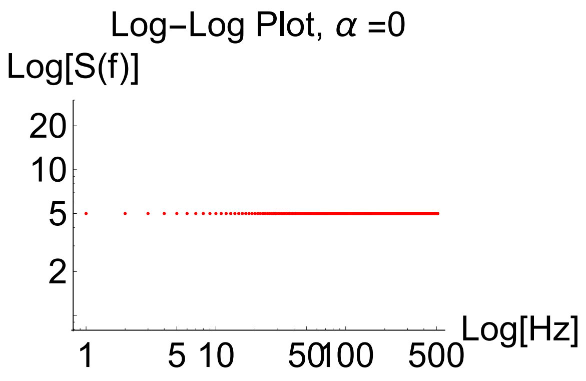

Reconstructed PSD, α =−1
S(f)
5
4
3
2
1
100 200 300 400 500
[Hz]

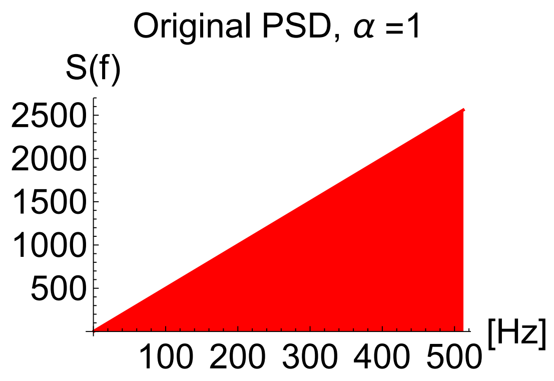

Log−Log Plot, α =−1
Log[S(f)]
5
1
0.50
0.10
0.05
10^{-2}
1
5 10
50 100
500
Log[Hz]

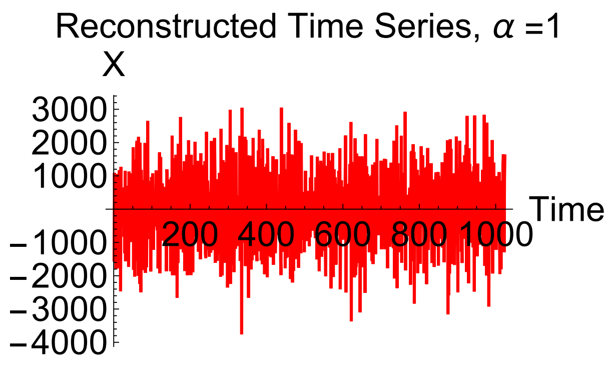

Original PSD, α =0
S(f)
10
8
6
4
2
100 200 300 400 500
[Hz]

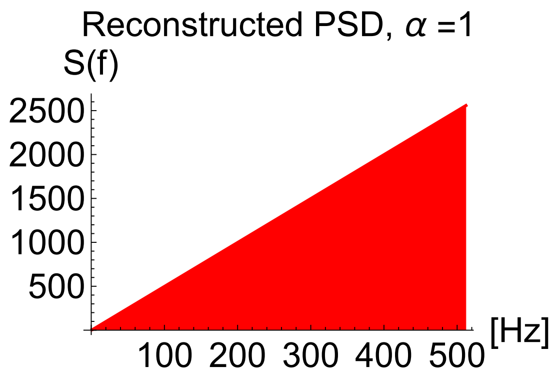

Reconstructed Time Series, α =0
X
200
100
−100
−200
200 400 600 800 1000
Time

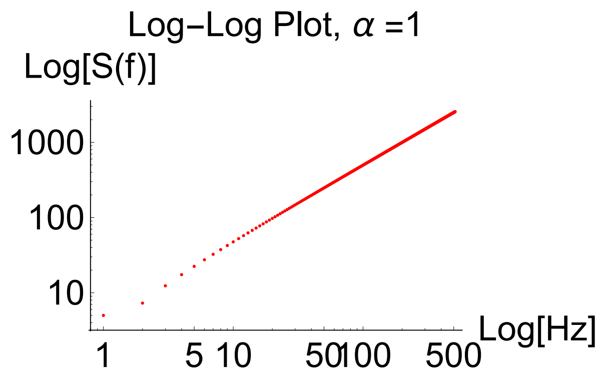

Reconstructed PSD, α =0
S(f)
10
8
6
4
2
100 200 300 400 500
[Hz]

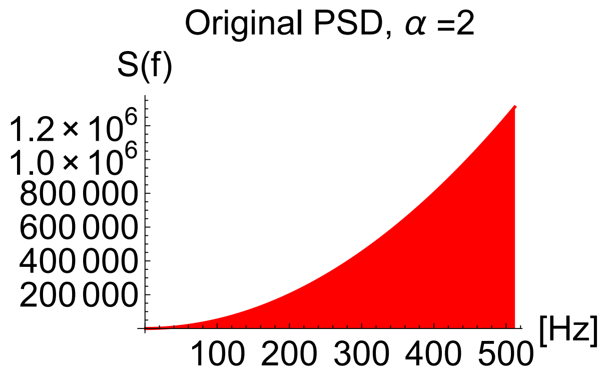

Log−Log Plot, α =0
Log[S(f)]
20
10
5
2
1 5 10 50100 500
Log[Hz]

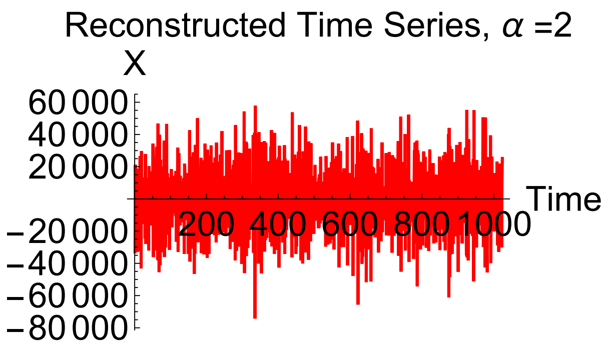

Original PSD, α =1
S(f)
2500
2000
1500
1000
500
100
200
300
400
500
[Hz]

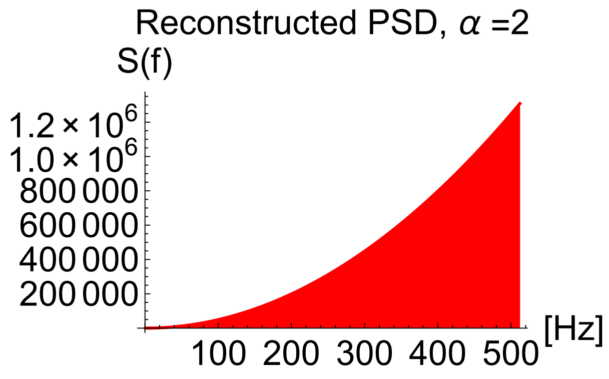

Reconstructed Time Series, α =1
X
3000
2000
1000
−1000
−2000
−3000
−4000
200
400
600
800
1000
Time

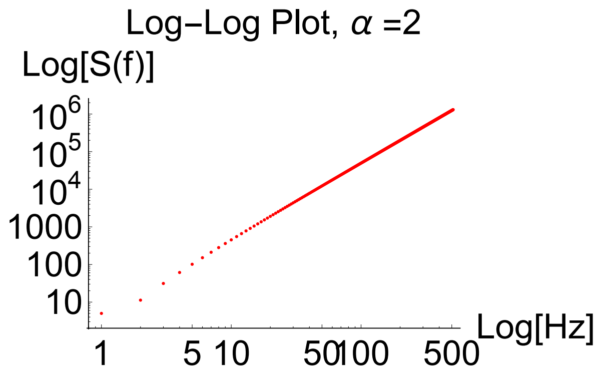

Reconstructed PSD, α =1
S(f)
2500
2000
1500
1000
500
100
200
300
400
500
[Hz]


Log−Log Plot, α =1
Log[S(f)]
1000
100
10
1
5
10
50
100
500
Log[Hz]


Original PSD, α =2
S(f)
1.2×10^6
1.0×10^6
800 000
600 000
400 000
200 000
100 200 300 400 500
[Hz]


Reconstructed Time Series, α =2
X
60 000
40 000
20 000
−20 000
−40 000
−60 000
−80 000
200 400 600 800 1000
Time


Reconstructed PSD, α =2
S(f)
1.2×10^6
1.0×10^6
800 000
600 000
400 000
200 000
100 200 300 400 500
[Hz]


Log−Log Plot, α =2
Log[S(f)]
10^6
10^5
10^4
1000
100
10
1 5 10 50 100 500
Log[Hz]