UNIVERSITY OF CALGARY

PHYSICS 581

COMPUTATIONAL PHYSICS III

Lab #2 & Assignment #2

Authors
Kunal Naidu 30020999
Ruand Hennawi 30024661

Instructor
Dr. Rachid Ouyed

June 5, 2023

Contents

Introduction

In this report, we will first be looking at noise reduction techniques using the spectral subtraction method. We will explore the effects of noise on signals with trends and how to recover the original signal from a noisy signal using Fourier Transform. Then, we will look at the Lorenz system. We will be exploring the Lorenz equations, which are non-linear differential equations that determine the evolution of a system described by three time-dependent state variables, and how to integrate them using the fourth-order Runge-Kutta scheme. We will also explore the concept of fixed points and how they are related to the Lorenz system. Finally, we will be applying the FFTs to real life situations.

Methods

Fourier Series

A Fourier series is a convenient representation of a periodic function (in time domain). It consists of a sum of sines and cosine terms. The trigonometric Fourier series of f(t) is given by

$$f(t) = a_0 + \sum_{n=1}^{\infty} (a_n \cos n\omega_0 t + b_n \sin n\omega_0 t) \quad (1)$$

where a_0, a_n, and b_n are the Fourier coefficients, and n is a positive integer representing the harmonic number. The sinusoid $\sin n\omega_0 t$ or $\cos n\omega_0 t$ is called the n^{th} harmonic of $f(t)$. If n is odd, the function is called the odd harmonic and if n is even, the function is called the even harmonic.

Moreover, Fourier Series in terms of frequencies is given by

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega_0 t + \sum_{n=1}^{\infty} b_n \sin n\omega_0 t \quad (2)$$

and the Fourier coefficients a_0, a_n, and b_n can be expressed as:

$$a_0 = \frac{2}{T} \int_0^T f(t) dt \quad (3)$$

$$a_n = \frac{2}{T} \int_0^T f(t) \cos n\omega_0 t dt \quad (4)$$

$$b_n = \frac{2}{T} \int_0^T f(t) \sin n\omega_0 t dt \quad (5)$$

An even extension of a function results in Fourier cosine series ($b_n \equiv 0$) and an odd extension of a function results in Fourier sine series ($a_n \equiv 0$).

Fourier Transform

A Fourier Transform builds on the basic idea of the Fourier Series. It allows us to describe periodic and non-periodic waves. Fourier transform allows us to analyze a function in terms of its frequency components. It no longer deals with integer multiples of the fundamental frequency, but instead, it deals with all frequencies, and therefore, one gets a continuous function of frequency.

The Fourier transform consists of the forward transform and the inverse transform. The forward transform takes a function of time signal and computes its frequency domain representation. It is given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \quad (6)$$

The inverse transform, on the other hand, takes a frequency representation and computes the time domain representation. It is given by:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \quad (7)$$

Discrete Fourier Transform

Discrete Fourier Transforms, DFT, are an extension of Fourier Transforms to the real world, since digital measurements cannot be taken continuously. The DFT converts a finite sequence of N complex numbers into another sequence of N complex numbers, which represent the frequency components of the original sequence. The DFT can be expressed mathematically as the following summation

$$X_k = < x, e^{-i2\pi k/N} >$$
$$= \sum_{n=0}^{N-1} x_k e^{-i2\pi k/N} = \sum_{n=0}^{N-1} x_k W_N^k \quad (8)$$

where x_k is the input sequence, X_k is the frequency domain representation of the sequence, and n and k are integers. The term W is known as the Twiddle Factor and is given by

$$W_N = e^{-i2\pi/N} \quad (9)$$

Fast Fourier Transform

The Fast Fourier Transform, FFT, is an algorithm that efficiently computes the Discrete Fourier Transform, DFT, of a sequence of N complex numbers or its inverse, IDFT. The FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. Then it calculates the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum.

The input to a typical FFT algorithm will consist of an array of data values, the number of time points (n), and the sample interval (dt). The simplest FFT algorithms require that the number of points in a data set be a power of two. Otherwise, one has to use padding in order for the number of points to be a power of two.

Noise and the FFT

One of the key applications of the FFT is noise reduction or filtering. The FFT can be used to analyze the frequency content of a signal and remove unwanted noise. A noisy signal $y(t)$ is a sum of the desired signal $x(t)$ and the noise $n(t)$ expressed as

$$y(t) = x(t) + n(t) \quad (10)$$

In the frequency domain, this may be denoted as

$$Y(f) = X(f) + N(F) \rightarrow X(f) = Y(f) - N(f) \quad (11)$$

where $Y(f)$, $X(f)$, $N(f)$ are Fourier transforms of $y(t)$, $x(t)$, $n(t)$, respectively. Moreover, the pure signal, denoted at $x(t)$, can be recovered by taking the IDFT of $X(f)$.

The Lorenz System

The lorenz system is described by non-linear differential equations. We defined the lorenz system using the following equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = xy - bz$$

Where x, y, and z are time dependant state variables and σ are real positive parameters. We will take $\sigma = 10$ and $b = \frac{8}{3}$.

Results and Discussion

Noise Reduction: The Spectral Subtraction Method

We took a simple signal given by

$$x(t) = \sin(2\pi f_0 t) + 0.5\sin(2\pi(2f_0)t) \quad (12)$$

with frequency $f_0 = 440$ Hz and sample it with $\Delta t = 22.2\mu s$. We took $N = 512$ intervals so that $t = i\Delta t$ with $i = (0, 511)$, and plotted the digitized signal. We obtained a the signal over many periods as seen in Figure 1.

We then computed an FFT using the function `np.fft.fft(xt)`. This function computes the one dimensional n-point DFT with the efficient FFT algorithm. We then calculated the power spectrum by taking the absolute value of the FFT to the power of two; `np.abs(fourier)**2`. Then, the frequency was found using `np.fft.fftfreq(N, delta_t)`, were `N` is the intervals (Window length) and `delta_t` is the time step, and plotted the power against the frequency. We got dominant frequencies of 440 Hz and 880 Hz as seen in Figure 1 since these are the frequencies present in the original signal as seen in equation ().

The signal was made noisy by adding a random number `r_i` between -0.5 and $+0.5$ to each `x_i`, where `x_i` is our signal, so that `y_i = x_i +r_i`, where `y_i` is our noisy signal. We then plotted the noisy signal for $i = (0, 511)$ and obtained the plot in Figure 2. One can see that it looks similar to the original signal obtained in Figure 1. We then ran an FFT on the noisy signal and plotted its power spectrum as seen in Figure 2. We can see the manifestation of the noise in the noisy signal plot but not the power spectrum as the power spectrum for the noisy signal looks the same as the power spectrum of the original signal. In the plot of the noisy signal, the noise appears as random fluctuations around the original sine wave. Moreover, the noise is not white noise as white noise will have a perfectly flat power spectrum, which is not the case here.

We found that the noise level was at 5% and so we zeroed out small components of the FFT below that noise level by multiplying the maximum power with 0.05, `np.max(power2) * 0.05`, and setting all frequency components below it to zero. We were able to recover the FFT of the two pure sine curves and got nearly the original signal `x_i` back as seen in Figure 2. And so, to completely scramble the original signal, we found that when `r_i` is between -5 and 5, it is large enough to do so.

Noise in Signals with Trends

We took another signal similar to the previous one over $N = 512$ bins so that $i = (0, 511)$ but this time with a trend function $g(i) = 1 + 0.025i$

$$x_i = 2\sin(\pi f_0 i) + \cos(\pi f_1 i) + g(i) \qquad (13)$$

where $f_0 = \frac{9}{512}$ and $f_1 = \frac{4}{512}$. We plotted the signal above for $i = (0, 511)$ as seen in Figure 3. Then the signal was made noisy by adding a random number `r_i` between -6 and $+6$ to each `x_i` so that `y_i = x_i + r_i`. We plotted the noisy signal as seen in Figure 4. The shape of the noisy signal is similar to the original signal, and so we can see that the original signal is visible in the noisy signal. The FFT of the noisy signal was then computed and its power spectrum was plotted as seen in Figure 4. The noise is not easily discernible, however, only the DC component was seen.

To recover the original signal, we zeroed out all frequencies below frequency magnitude of 180 and then computed an IDFT. As seen from Figure 4, the original signal was not recovered.

So in order to recover the original signal, we will remove the trend from the noisy signal and redo the analysis above. Then we will zero out all frequencies below frequency magnitude of 9 and compute an IDFT and then add the trend.

We plotted the detrended noisy signal as seen in Figure 5, and we saw that the noise was easily discernible, and were not able to see the main components of the noiseless signal as seen. And then after zeroing out all frequencies below frequency magnitude of 9 and computing an IDFT and then add the trend $g(i)$, we recovered the original signal as seen in Figure 6.

Can you think of a way of using what you have learned to encrypt a signal (or message) and only you or whoever you chose to can recover (i.e. access) the original signal (or hidden info).

So, when dealing with noise when it comes to signals with trends, it's important to take the trend into account when filtering out the noise. Ignoring the trend can lead to inaccurate results. If trends are ignored in signals while filtering the noise, there is a risk of misinterpreting the data. For example, in financial series, trends can indicate important market trends that can be used for analysis. If the trend is removed along with the noise, important information can be misinterpreted. One way to encrypt a signal or message is to use a technique called steganography. Steganography is the practice of concealing information within another message or physical object to avoid detection. For example, education and business institutions, intelligence agencies, the military, and certified ethical hackers use steganography to embed confidential messages and information in plain sight. The messages are embedded in the signal by adding noise to certain frequencies, which would not be noticeable to someone who doesn't know the algorithm used to embed the message.

The Lorenz System

First we started by finding the fixed points for the system. These fixed points would be where the system would be stable. We did this by solving for the lorenz equations where all the derivatives above are equal to 0. By solving these equations we can get the fixed points x_{eq}, y_{eq}, and z_{eq}.

$$0 = \sigma(y_{eq} - x_{eq}) \tag{14}$$

$$0 = rx_{eq} - y_{eq} - x_{eq}z_{eq} \tag{15}$$

$$0 = x_{eq}y_{eq} - bz_{eq} \tag{16}$$

We then took equation 14 and simplified it to

$$y_{eq} = x_{eq} \tag{17}$$

We then took equation 16 and simplified it to

$$z_{eq} = \frac{x_{eq}^2}{b} \tag{18}$$

Then we took equation 15 and got

$$x_{eq} = \sqrt{r-1} \tag{19}$$

Using these three equations we solved for the fixed points to be

$$x_{eq} = \sqrt{r-1} \tag{20}$$

$$y_{eq} = \sqrt{r-1} \tag{21}$$

$$z_{eq} = \frac{r-1}{b} \tag{22}$$

Next, we integrated the Lorenz equations in the linear regime using the fourth-order Runge-Kutta scheme. We used an initial of $(t_0, x_0, y_0, z_0) = (0, 5, 5, 5)$ and $r = 28$. After changing the number of integration points and the time interval we noticed at 2000 integration's points and stepping time by 0.01 that we found the strange behavior of the solution seen in figure 7. We see that this strange behavior looks like two spirals that connect to each other. The overall shape does look similar to that of a butterfly.

Looking at the animation of the position of the system over time, we were able to locate the fixed point and at least to the time length that was shown in the animation that the fixed points are not visited by the system. If they were visited by the system then there would be no more change in the system after reaching those points.

Figure 7 shows us the final solution to the lorenz equations as an orbit in phase space. We again see this spiral shape and over time. We noticed that the beginning time values are traced closely with the later time values.

We next looked at $r = 28$ and we varied the initial conditions for the initial conditions

$$\begin{aligned}(x_0, y_0, z_0) = {} & (0,0,0), (0.1, 0.1, 0.1), \\ & (0,0,20), (100,100,100), \\ & (8.5, 8.5, 27), (5,5,5)\end{aligned}$$

We then integrated the system using these conditions to get the figure 8. For the initial condition $(5,5,5)$ we see that our solution did not intercept our fixed point and made the butterfly like shape that we previous talked about. For the initial condition $(0,0,0)$ we see that the solution just stays at the initial position over time. This is because another solution to the fixed points that we previously calculated is $(0,0,0)$. Thus the initial condition was a state where the x, y and z do not change over time. Looking at the initial condition $(0.1, 0.1, 0.1)$ we see that it has similar shape to $(5,5,5)$. For the initial condition $(0,0,20)$ we see that the solution goes towards $(0,0,0)$ and then doesn't change. For initial Condition $(100,100,100)$, it looks like the solution started to approach infinity within the time series that we used. When looking at the initial condition $(8.5, 8.5, 8.5)$ we see that the solution looks similar to that of (5,5,5), but that the right spiral is more heavily biased than the left spiral.

We then looked at the initial condition $(5,5,5)$ and looked at the values R = 14.6, 14.75, 60, 90 and again solved the lorenz equations to get the following figure 9. After solving for various r values, we were able to find that the system makes a transit from a singular spiral behaviour to a dual spiral behavior at approximately r = 14.75.

We then varied r from 0 to 30 and plotted x to get the bifurcation diagram of the lorenz system (figure 10). The bifurcation diagram shows us that we transitioned from a singular fixed point to two fixed points. Approximately at r = 1 we have 0 fixed points that transition into two fixed points and at r = 24.74 we see that the system has destablized. An example of global bifurcation has been observed experimentally in turbulent Rayleigh-Benard convection[1].

The Fourier Analysis

We then looked at Fourier analyzing the Lorenz equations for a range of r values (r=1,3,13,14.5,26). We only looked at the solution to the x variable. We calculated the phase-space plot, the solution of x and the spectra. Looking at figure 11 we see that on the left for r=1, we have a very steep log like graph. we see that for the phase space plot we have a exponential like increase and that for the spectra we have a polynomial like increase in our amplitude against frequency. When we increase r to 3 we see that we get something similar to a $sin(t)/t$ graph and that in our phase space plot we are starting to see a spiral like shape. In our spectra we see our dc value and two small peaks as on either side of it. When we go to r=13 we start to see much more of a sinusoidal shape and the phase space plot takes more of a spiral shape where the middle of the plot is more dense. The spectra looks similar to the previous but the two peaks mentioned before are larger. Going to r=14.5, we see that the x(t) plot is as the sinusoidal plot goes in time, the decrease in the amplitude itself is decreasing over time(the amplitude is larger over time that of the previous r). We see that the phase space plot has a very dense spiral compared to previous. We again see that the spectra has two peaks that are larger than before. When we go to r=26 we see that the x(t) plot has changed. It is still cyclical wave but the shape of the wave is much more sporadic. When looking at the phase space plot we see that now we have two spirals that intertwine with each other. Looking at the spectra we see that there are several different peaks with large amplitudes.

We then looked at the qualitative picture of the variation of the spectra over a range of r. We used a spectral density plot. We noticed that for r approximately equal to 1 that we have a large amount of frequencies that are small respectively. We notice that there is a transition around r = 3 where the amplitudes become very small for every frequency. Then there is another transition around r=25 where we have several different amplitudes of various amplitudes around frequency 0.

FFT: Applications

1 A Stellar Binary System

Here, we plotted the light curve in Figure 13, along with its amplitude spectrum after computing the FFT of the given time-series. The light curve plot shows the count rate of photons as a function of time for a binary system consisting of a neutron star (the pulsar) and a companion star. The amplitude spectrum plot shows the amplitude of the Fourier transform of the count rate time-series as a function of frequency. From the amplitude spectrum, we were able to find periodic signals at frequency 2.5 Hz and 4 Hz.

2 Heart Beats

Here, using the given data in the text file, we checked if the length of the data points was a power of two, and found that it is, and therefore, no padding was necessary. We computed an FFT code on the sequence of heart beats and plotted the amplitude spectrum in Figure 14. When we excluded the DC component, we found that the two most dominant frequencies were 1.98364258 Hz and 2.01416016 Hz with periods 0.50412308 s and 0.49648485 s. A normal resting heart rate for adults ranges from 60 to 100beats per minute which is typically between 0.8 to 1.2 beats per second. So the periods do not make sense as they don't fall within the range of 0.8 to 1.2 beats per second.

3 Solar Activity

We plotted the number of sunspots as a function of time as seen in Figure 15 and got an estimate of the length of the cycle of approximately 2.075 months. Using the given data in the text file, we checked if the length of the data points was a power of two, and found that it is not, and therefore, padding was necessary.

After padding the data, we computed an FFT on the sequence and then made a graph of the magnitude squared of the Fourier coefficients, $|c_k|^2$ as a function of k (the power spectrum) as seen in Figure 16. We see a DC component and a peak around $k = 3600$.

4 Application to Financial Series

We then looked at price series data for the trading index and stock prices for six different companies(figure 17). We notice that SANDP has gradually increasing stock prices, FORD has a steady stock price which greatly increases in price and then greatly decreases in price back to approximately the initial price. We see that GM has a lot of variation between high and low stock prices. Microsoft has steadily increasing stock prices. SUN has exponentially increasing stock prices similarly to USTB3m.

We then looked at the continuously compounded returns $R_i = ln(P_i/P_{i-1})$. Looking at figure 18, we see that for all the companies the returns vary plus minus 0.2.

We then looked at the auto-correlation for each of the companies in figure 19 we noticed for all of the companies the correlation values decreases over k and that there is non randomness as k increases.

We then looked at the at the power spectrum of the stock prices of the different companies. SANDP, Microsoft only have the DC peak while, Ford, Sun, and USTB3M have comparatively significant non zero values that are around the 0 frequency. GM has very small peaks around 0 frequency.

We then looked at the Dow Jones Industrial Average, which is a measure of the average prices on the US stock market. We looked at the final price at which a stock is traded on a given trading day for each business day from late 2006 to the end of

2010. We plotting the price of the stock over time and its corresponding power spectrum in figure 21. We see that the price decreases until approximately arbitrary time unit 600 and then it start increasing. We notice that the Power Spectrum only has one peak at 0 frequency.

We then looked at the auto-correlation function for the data in figure 22, we again notice the similar shape as the previous stock prices. We see a non randomness in k values as k increases.

5 Global Warming

We plotted the time series of CO_2 concentration, showing the monthly variations, as seen in Figure 23. We can see that the trend of the CO2 concentration over the years is linear which means that its showing a steady increase over time.

After Fourier analyzing the seasonal oscillation, the linearly increasing trend was subtracted from the data obtaining the plot in Figure 24. Then, we generated a power spectrum seeing peak at around The peak at a frequency of approximately 1 in Figure 25 which corresponds to the seasonal oscillation in the data. We can also see a very smaller peak at another frequency, which represent other cycle present in the data.

If the current trends continue, the concentration will reach toxic levels when the CO_2 concentration reaches 70000 as seen in Figure 26. So we used the linearly increasing trend that was subtracted from the data above and estimated that it will reach the toxicity level in year 50000, and so the linear increase is accelerating.

Conclusion

In conclusion, we explored noise reduction techniques using the spectral subtraction method and applied them to signals with trends. The effects of noise on signals were investigated, and the original signal was recovered from a noisy signal using Fourier Transform and discovered that it is possible to reduce the impact of noise and improve the quality of data. Moreover, the Lorenz system and its non-linear differential equations were explored, along with the concept of fixed points and their relationship to the Lorenz system. We have explored the Lorenz system by studying its fixed points and their stability properties. We also studied the integration of the Lorenz equations using the fourth order Runge-Kutta. We found that this method is a reliable way to solve the Lorenz equations numerically and obtain accurate solutions. Finally, the Fast Fourier Transform was applied to real life situations.

References

[1] University of Calgary. Department of Physics and Astronomy. *Phys581-FFT-Lab-Assign-2023*. Personal Communication. 2023.

Appendix

Plots

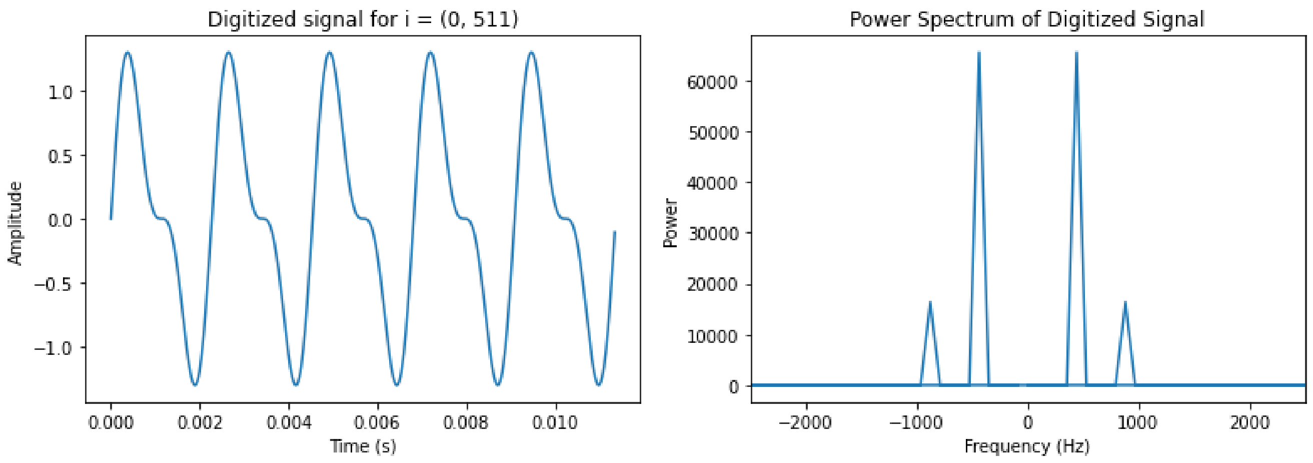

Figure 1: Signal for $x(t) = \sin(2\pi f_0 t) + 0.5\sin(2\pi(2f_0)t)$, and its power spectrum.

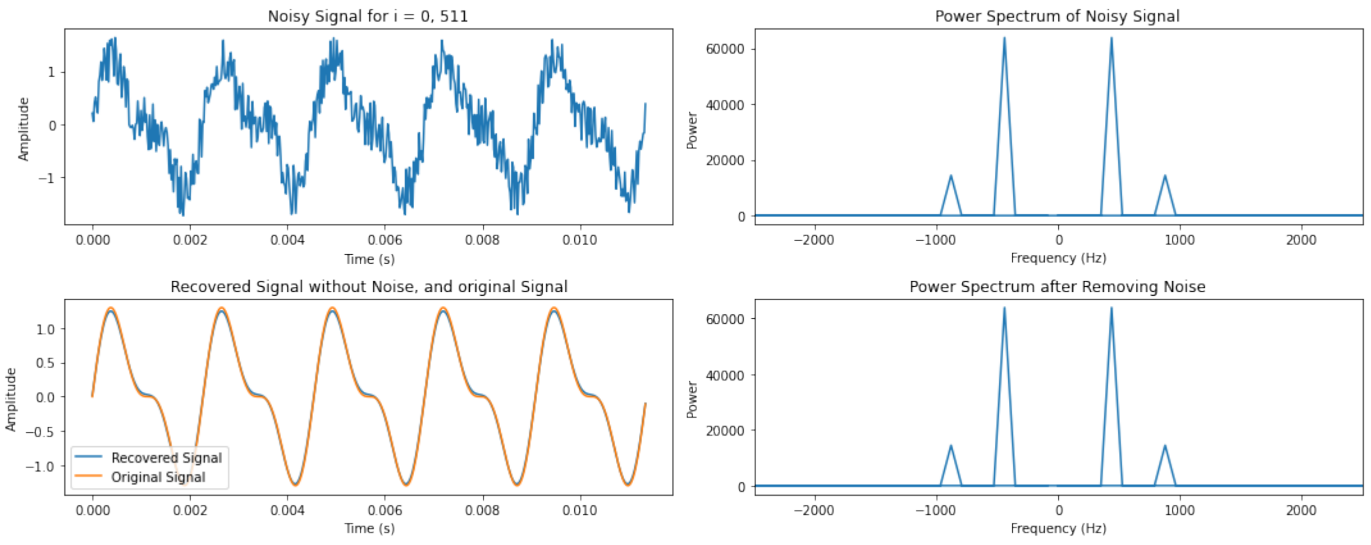

Figure 2: Noisy signal for $x(t) = \sin(2\pi f_0 t) + 0.5\sin(2\pi(2f_0)t)$ and its power spectrum, recovered signal and its power spectrum.

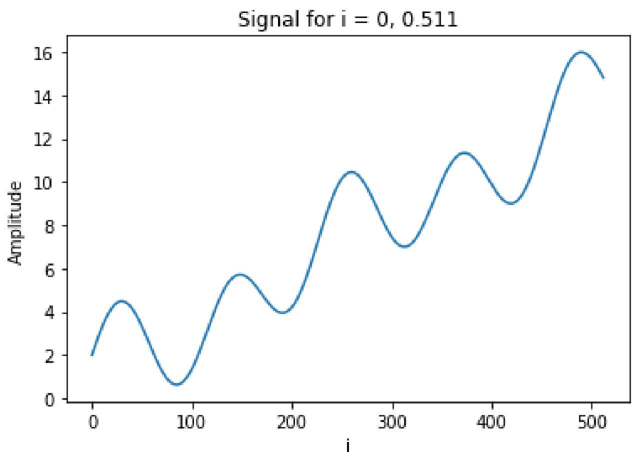

Figure 3: Signal for $x_i = 2\sin(\pi f_0 i) + \cos(\pi f_1 i) + g(i)$

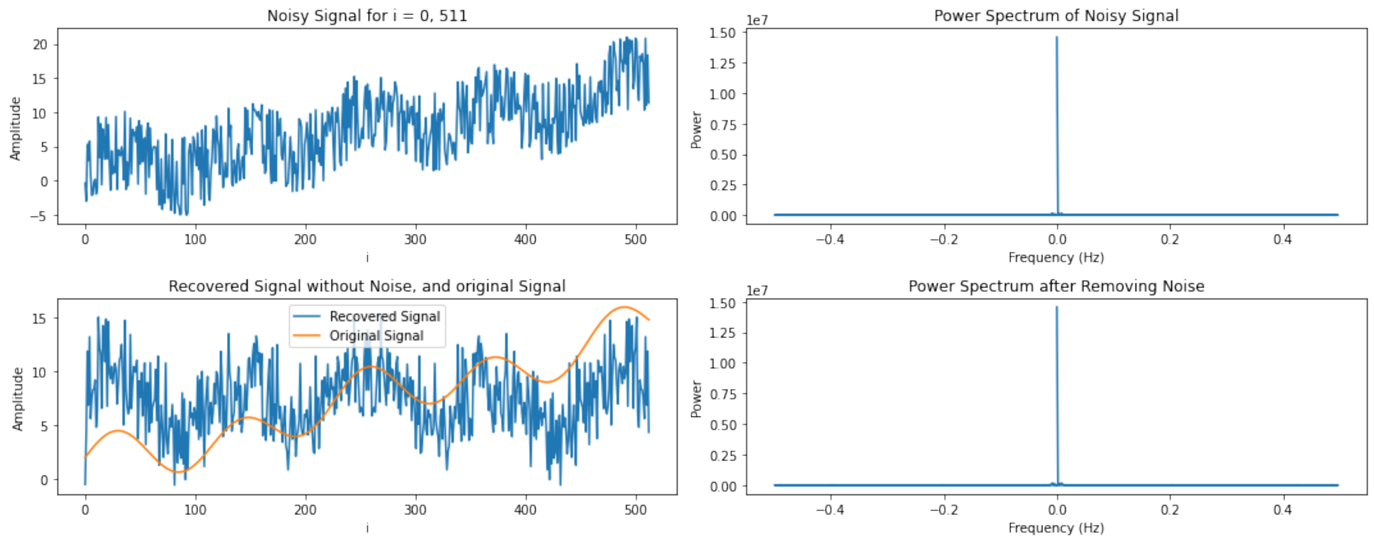

Figure 4: Noisy signal for $x_i = 2\sin(\pi f_0 i) + \cos(\pi f_1 i) + g(i)$ and its power spectrum, recovered signal and its power spectrum.

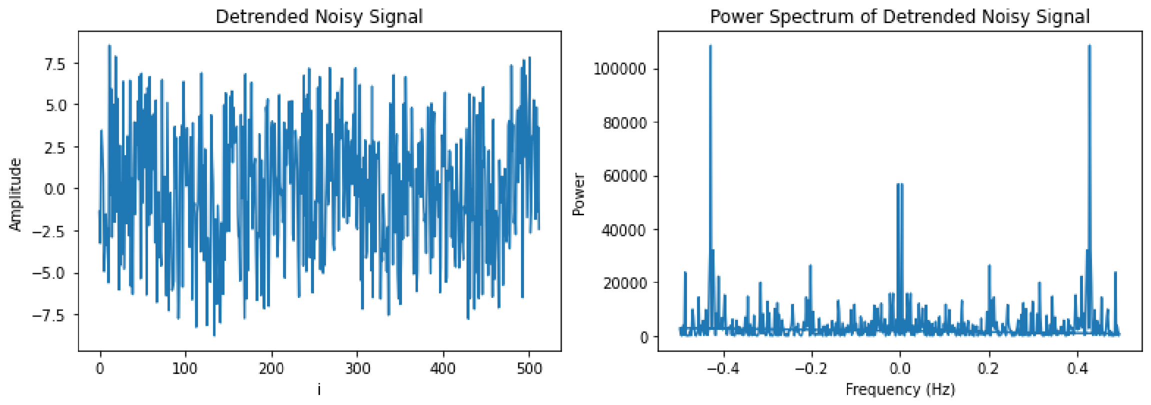

Figure 5: Detrended Noisy signal for $x_i = 2\sin(\pi f_0 i) + \cos(\pi f_1 i) + g(i)$ and its power spectrum.

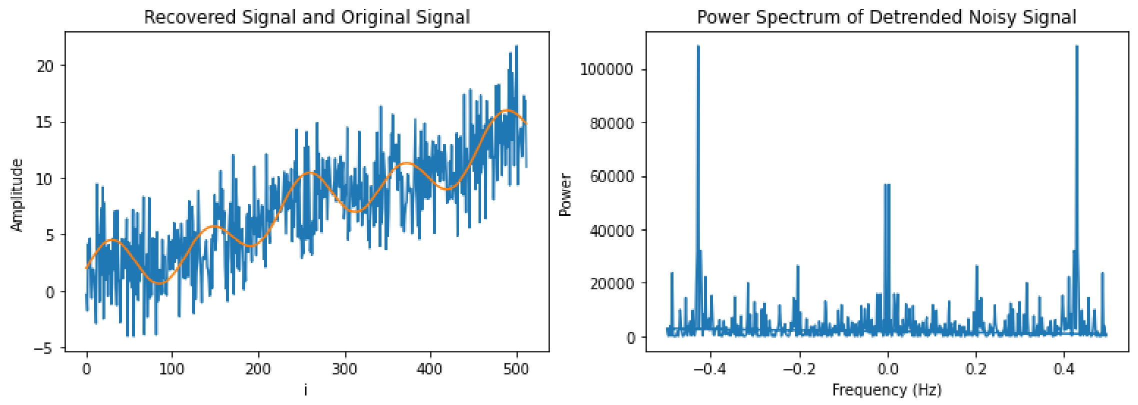

Figure 6: Recovered signal and its power spectrum.

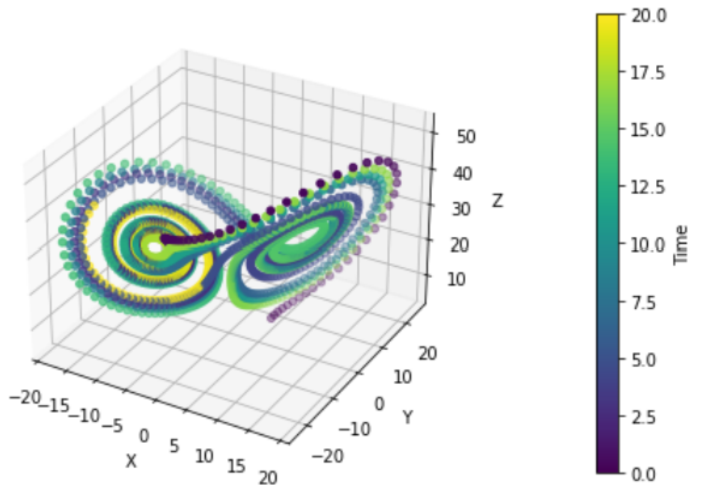

Figure 7: Solution to the lorenz equations using initial values $(t, x_0, y_0, z_0) = (0, 5, 5, 5$ and $r = 28$. This strange behavior was found with 2000 integration points and time step size of 0.01

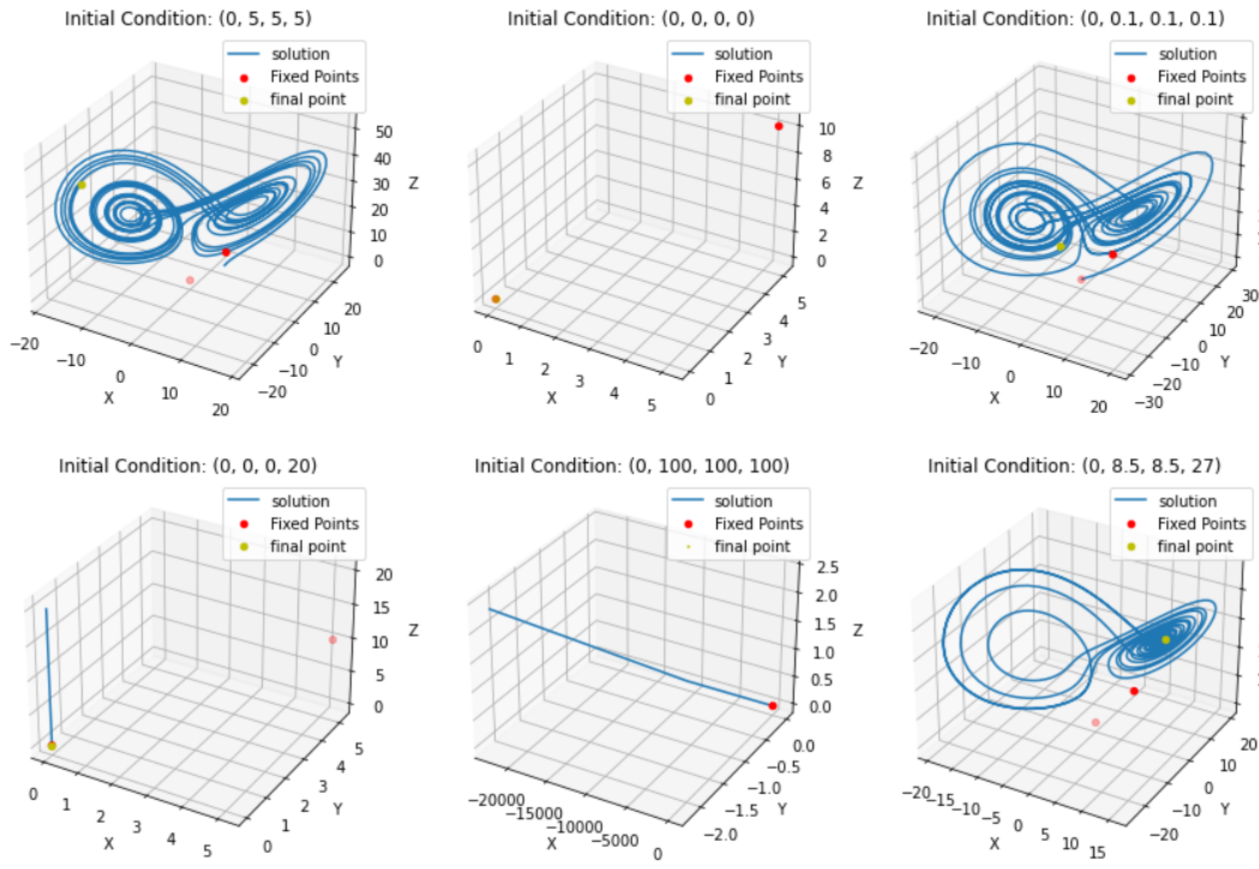

Figure 8: Solution for the lorenz equations using different initial conditions the initial conditions are formatted by (t_0, x_0, y_0, z_0). The fixed point and the final point are shown. The final point is the final position in the time series.

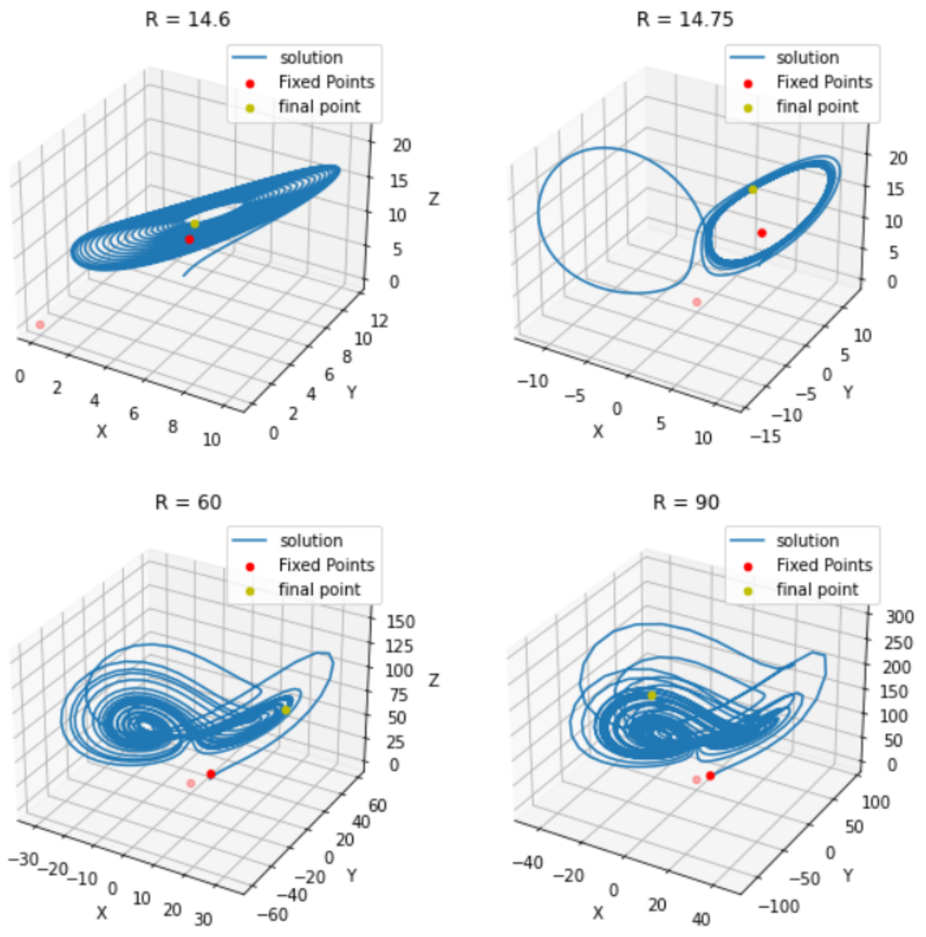

Figure 9: Solution for the lorenz equations for the initial conditions x=5, y=5, z=5 for different r values. The fixed point and the final point are shown. The final point is the final position in the time series.

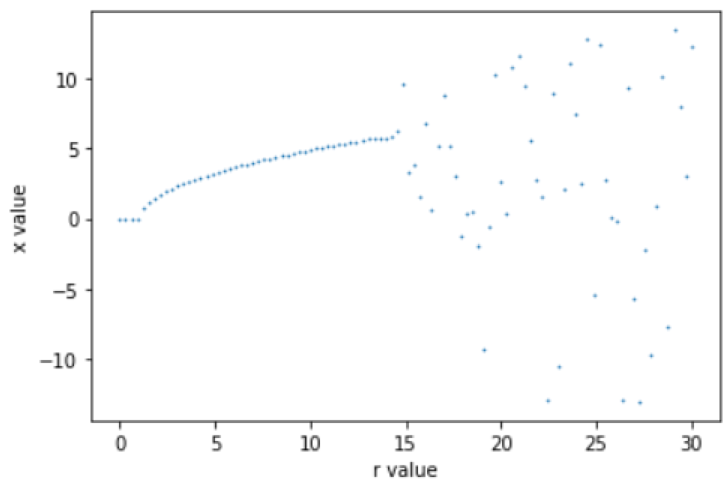

Figure 10: bifurcation diagram of the lorenz system for the value r.

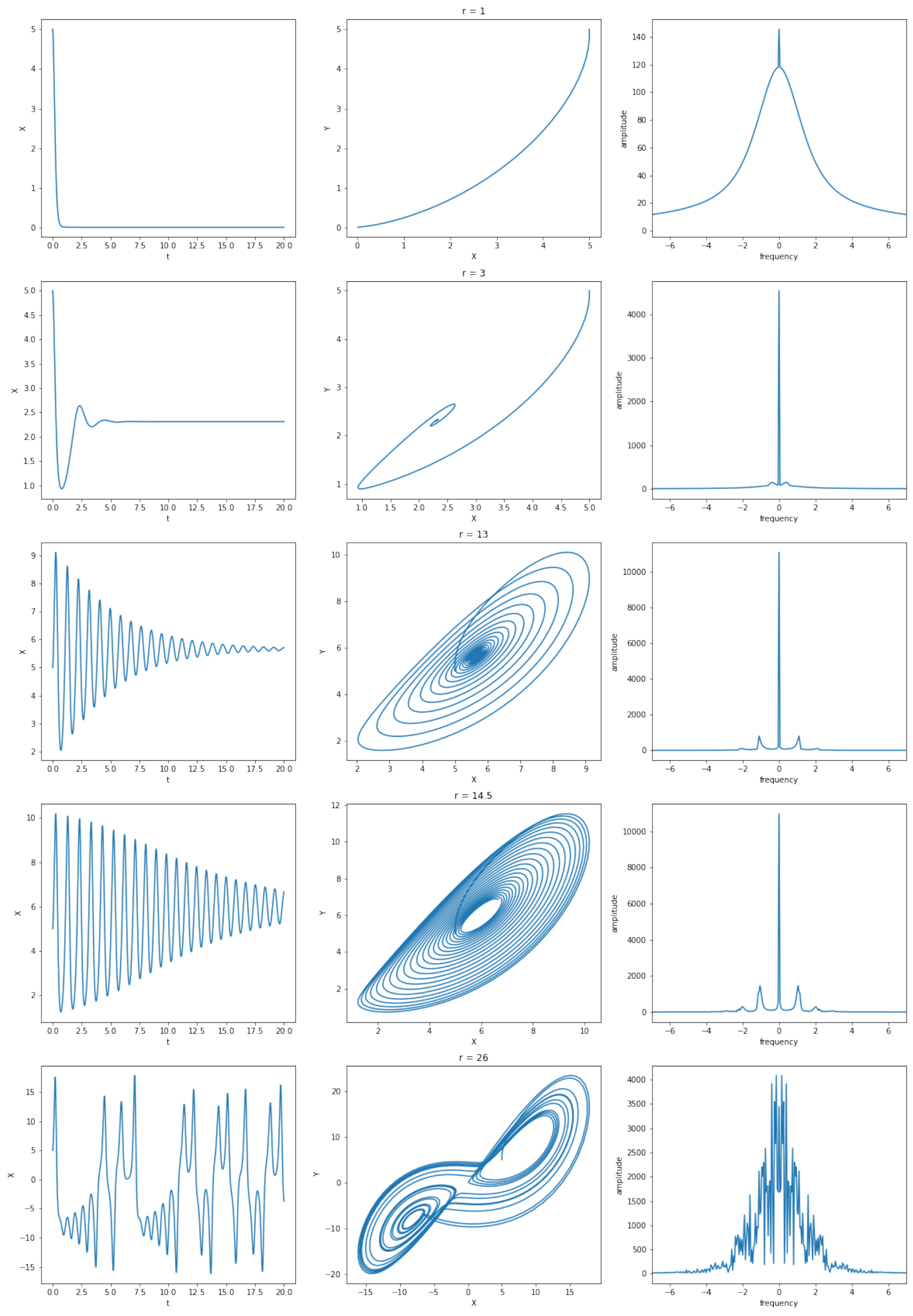

Figure 11: solution to the lorenz equations on the left is plots of x(t), the middle plot is the phase-space plot and the right plot is the spectra for different r values.

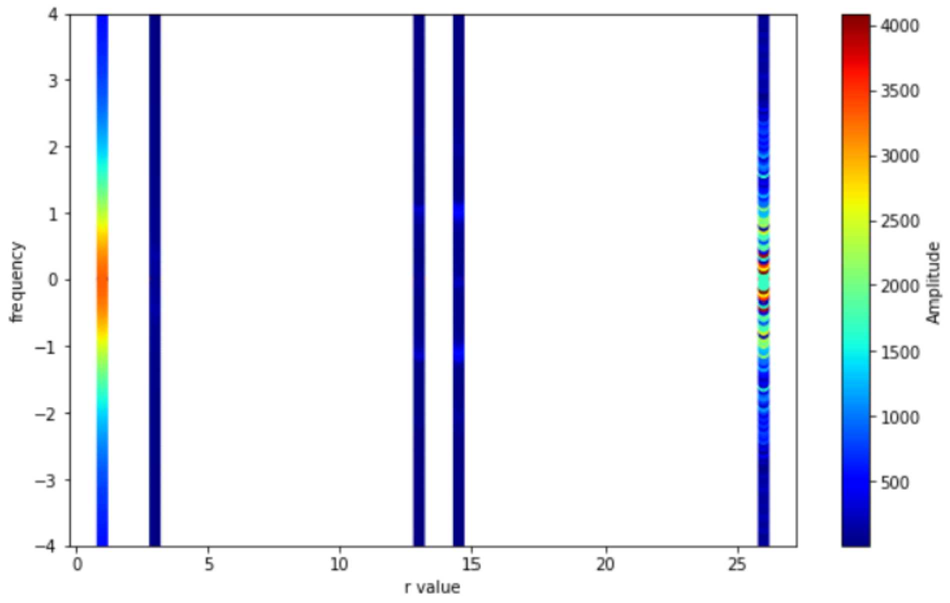

Figure 12: spectral density plot for the solution of the lorenz equations for the different values of r

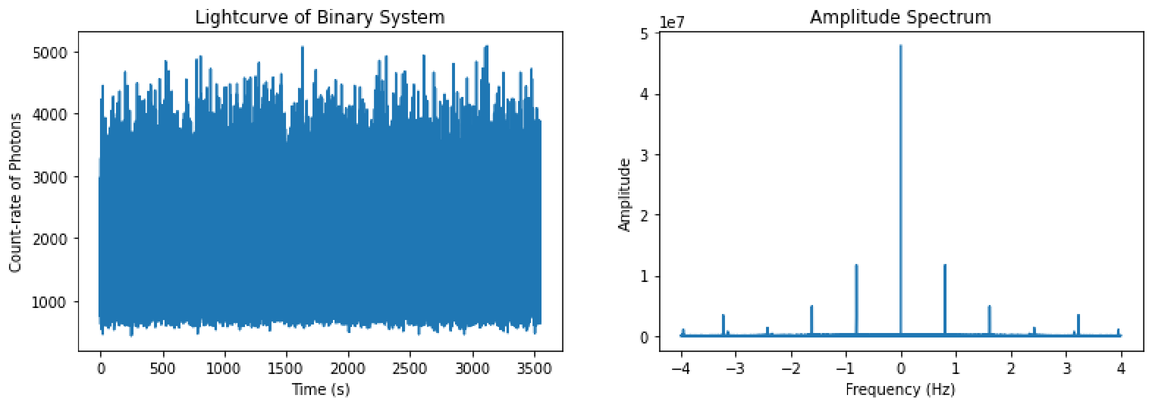

Figure 13: Count-rate as function of time

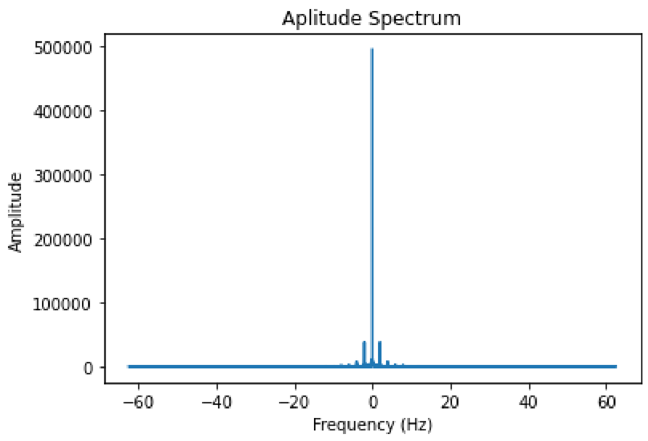

Figure 14: Amplitude Spectrum of Heart Beats

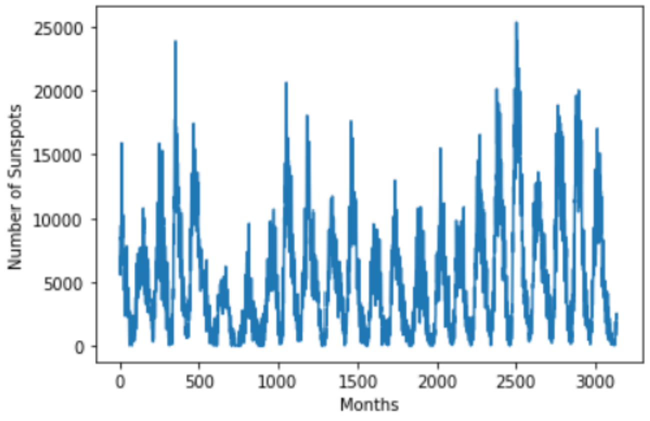

Figure 15: Number of sunspots as a function of time in months.

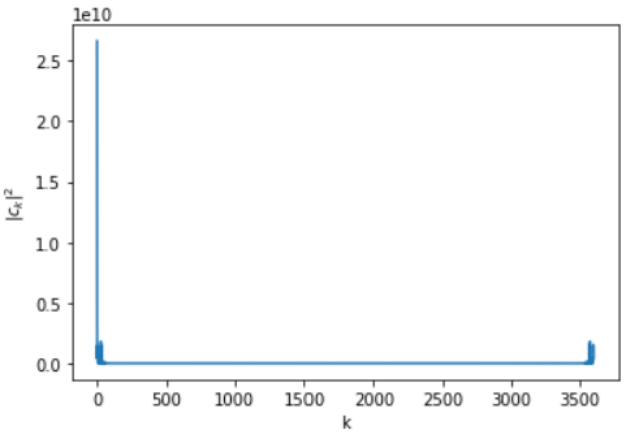

Figure 16: Power spectrum for sunspot signal

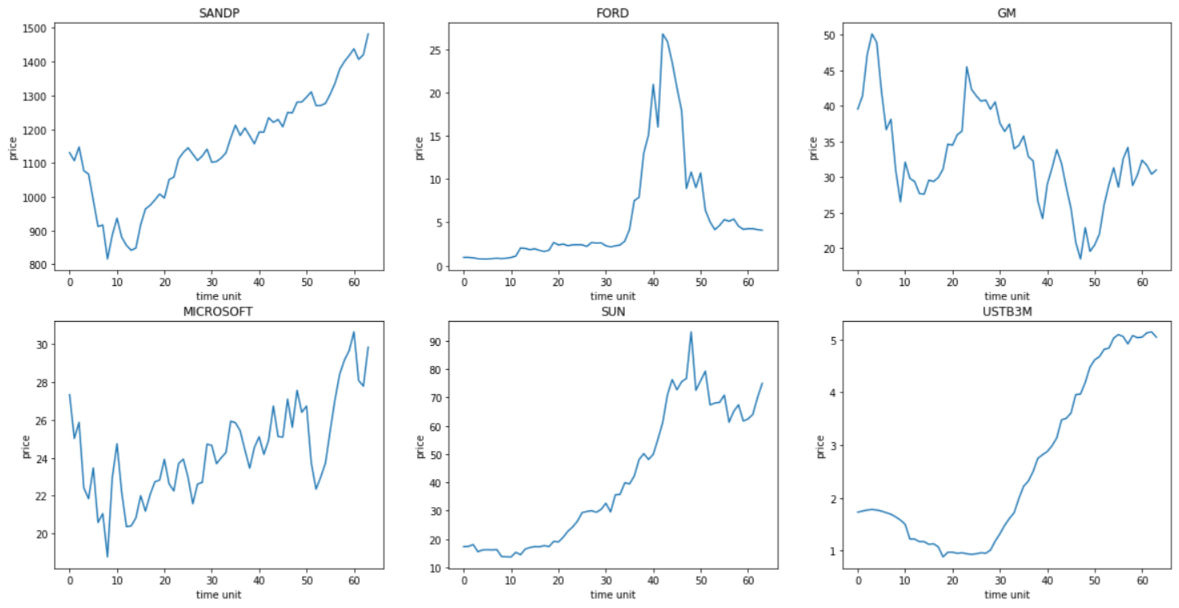

Figure 17: Stock price series for the trading index and stock prices for six companies.

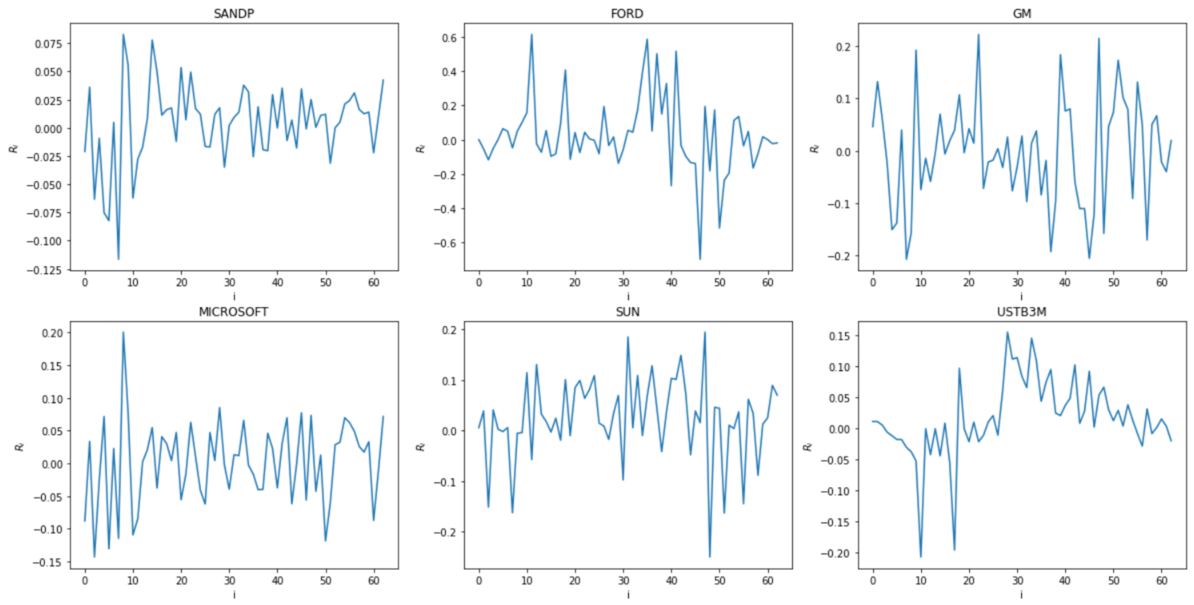

Figure 18: Continuously compounded returns for stock prices for six different companies.

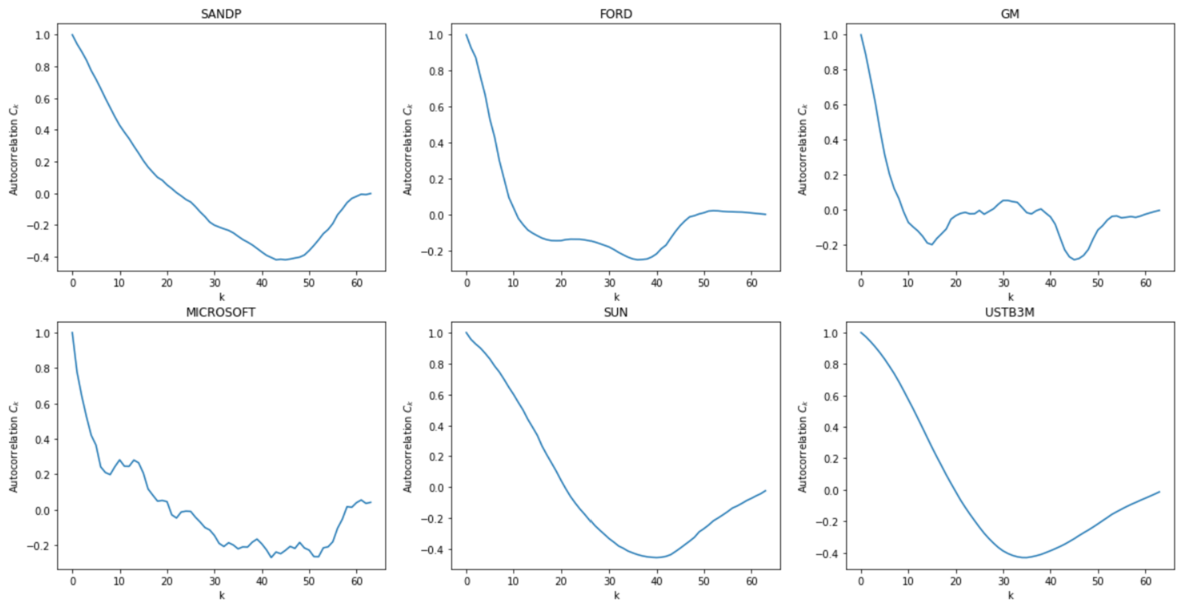

Figure 19: Auto-correlation plots for the different companies for their stock prices over time.

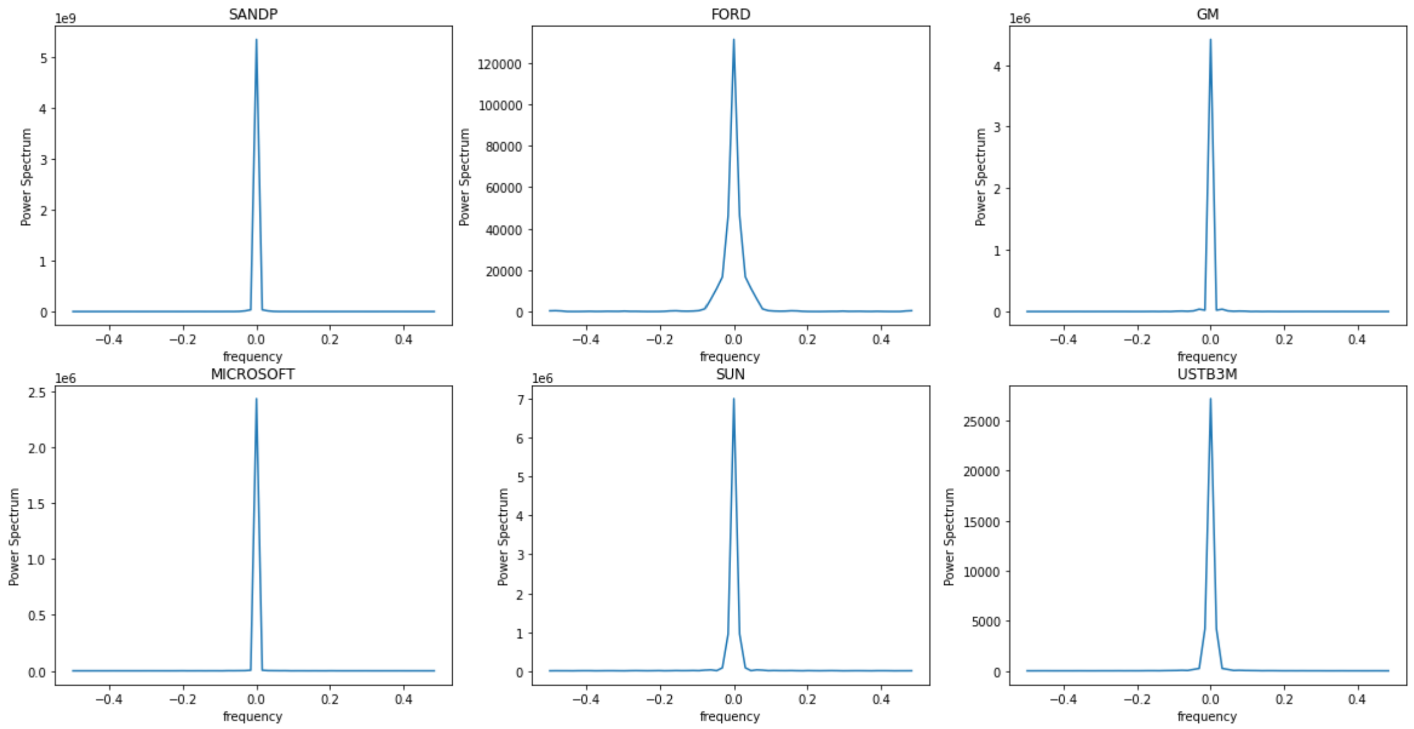

Figure 20: The power spectrum for the stock prices for the different companies over time

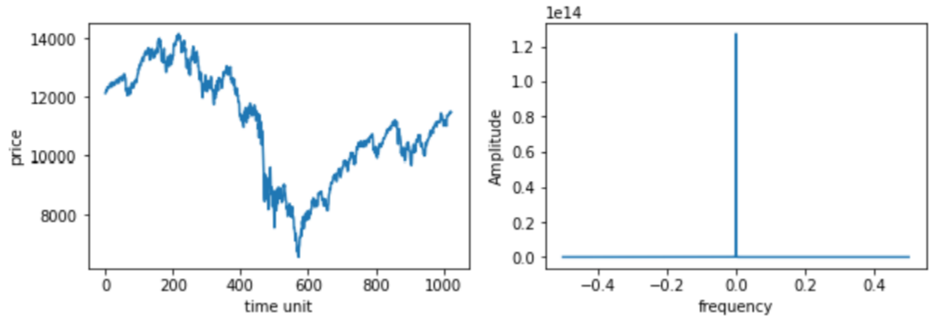

Figure 21: On the left is the daily closing value for each business day from late 2006 until the end of 2010 of the Dow Jones Industrial Average. On the right is the power spectrum.

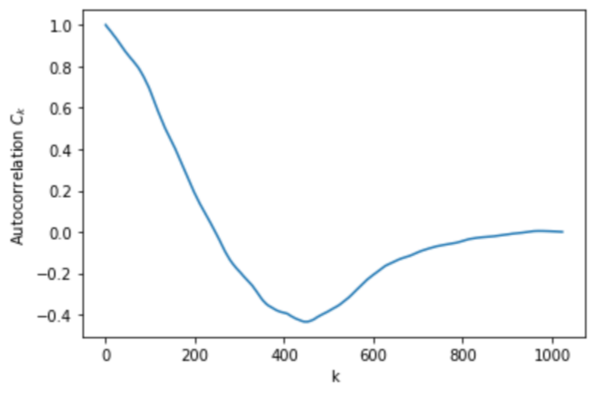

Figure 22: Auto correlation function for the Dow Jones Industrial Average from late 2006 to end of 2010.

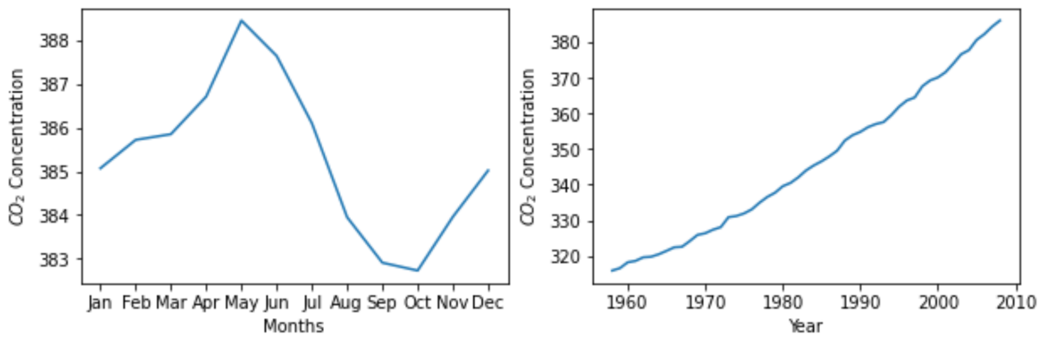

Figure 23: Time series of CO_2 concentration, showing the monthly variations, and over the years.

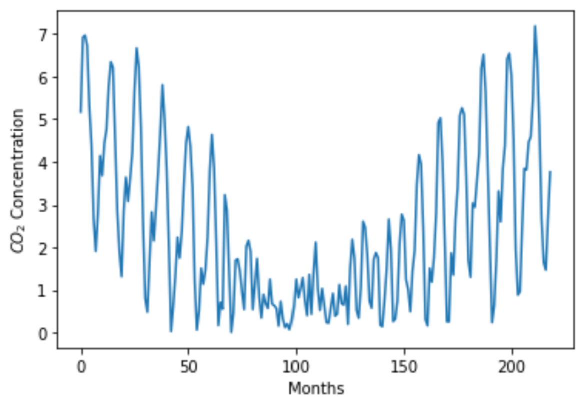

Figure 24: CO_2 concentration against months

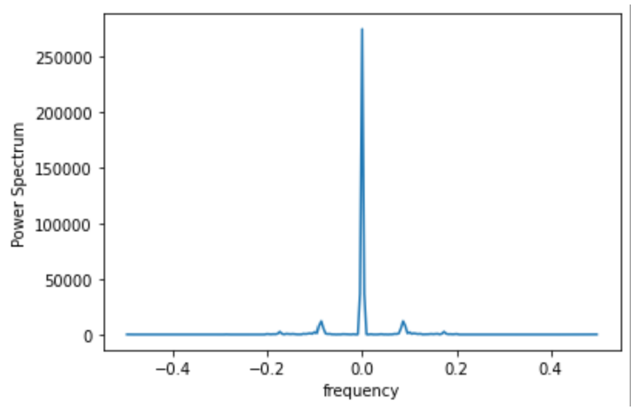

Figure 25: Power Spectrum

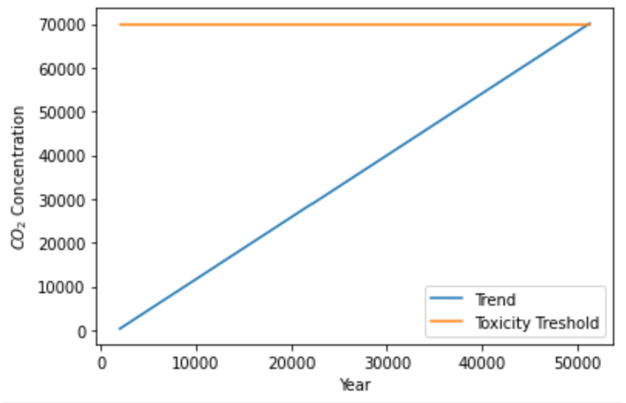

Figure 26: CO_2 concentration against years

Code

```
# -*- coding: utf-8 -*-
"""Lab2
```

```
Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1RHKYFCuR_gy2CFLEEsF555kWEScnbcVJ
"""

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import PillowWriter
from scipy.fft import fft, fftfreq
from mpl_toolkits import mplot3d
import pandas as pd

def simple_signal(f_0, t):
  return np.sin(2 * np.pi * f_0 * t) + 0.5 * np.sin(2 * np.pi * (2 * f_0) * t)

f_0 = 440
delta_t = 22.2e-6
N = 512
t = np.arange(0, 22.2e-6*512, 22.2e-6)
xt = simple_signal(f_0, t)

#running an FFT
fourier = np.fft.fft(xt)

#calculating the power spectrum
power = np.abs(fourier)**2
freq = np.fft.fftfreq(N, delta_t)

#ceate subplots
fig, axes = plt.subplots(ncols=2, figsize=(13, 4))

#plotting the signal
axes[0].plot(t, xt)
axes[0].set_title('Digitized signal for i = (0, 511)')
axes[0].set_xlabel('Time (s)')
axes[0].set_ylabel('Amplitude')

#plotting the power spectrum
axes[1].plot(freq, power)
axes[1].set_title('Power Spectrum of Digitized Signal')
axes[1].set_xlim([-2500, 2500])
```

```
axes[1].set_xlabel('Frequency (Hz)')
axes[1].set_ylabel('Power')

plt.show()

r_i = np.random.uniform(-0.5, 0.5, size = N)
y_i = xt + r_i

#run the FFT on the noisy signal
fourier2 = np.fft.fft(y_i)
freq2 = np.fft.fftfreq(N, delta_t)
power2 = np.abs(fourier2)**2

#determined noise level (5%) and set all frequency components below it to zero
noise_level = np.max(power2) * 0.05
fourier2[np.where(power2 < noise_level)] = 0

#run the inverse FFT to recover the signal without noise
y_cleaned = np.fft.ifft(fourier2).real

#power spectrum after removing noise
power_cleaned = np.abs(fourier2)**2

#ceate subplots
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(15, 6))

#plotting the noisy signal
axes[0, 0].plot(t, y_i)
axes[0, 0].set_title('Noisy Signal for i = 0, 511')
axes[0, 0].set_xlabel('Time (s)')
axes[0, 0].set_ylabel('Amplitude')

#plotting the power spectrum for noisy signal
axes[0, 1].plot(freq2, power2)
axes[0, 1].set_title('Power Spectrum of Noisy Signal')
axes[0, 1].set_xlim([-2500, 2500])
axes[0, 1].set_xlabel('Frequency (Hz)')
axes[0, 1].set_ylabel('Power')

#plot the cleaned signal and compare to original signal
axes[1, 0].plot(t, y_cleaned, label='Recovered Signal')
axes[1, 0].plot(t, xt, label='Original Signal')
axes[1, 0].set_title('Recovered Signal without Noise, and original Signal')
```

```
axes[1, 0].set_xlabel('Time (s)')
axes[1, 0].set_ylabel('Amplitude')
axes[1, 0].legend()

#plot the power spectrum after removing noise
axes[1, 1].plot(freq2, power_cleaned)
axes[1, 1].set_title('Power Spectrum after Removing Noise')
axes[1, 1].set_xlim([-2500, 2500])
axes[1, 1].set_xlabel('Frequency (Hz)')
axes[1, 1].set_ylabel('Power')

fig.tight_layout()
plt.show()

def similar_signal(f_0, f_1, i, g):
  return 2 * np.sin(np.pi * f_0 * i) + np.cos(np.pi * f_1 * i) + g

def g(i):
  return 1 + 0.025 * i

f_0 = 9/512
f_1 = 4/512
N = 512

i = np.linspace(0, 512, 512)

x_i = similar_signal(f_0, f_1, i, g(i))

#plot the similar signal
plt.plot(i, x_i)
plt.xlabel("i")
plt.ylabel("Amplitude")
plt.title("Signal for i = 0, 0.511")
plt.show()

r_i = np.random.uniform(-6, 6, size = N)
y_i = x_i + r_i

#run the FFT on the noisy signal
fourier3 = np.fft.fft(y_i).real
freq3 = scipy.fft.fftfreq(len(i), d=i[1]-i[0])
power3 = np.abs(fourier3)**2
```

```
#set all frequency components below 180 to zero
fourier3[np.where(power3 < 180)] = 0

#run the inverse FFT to recover the signal without noise
y_cleaned = np.fft.ifft(fourier3).real

#plot the power spectrum after removing noise
power_cleaned = np.abs(fourier3)**2

#ceate subplots
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(15, 6))

#plotting the noisy signal
axes[0, 0].plot(i, y_i)
axes[0, 0].set_title('Noisy Signal for i = 0, 511')
axes[0, 0].set_xlabel('i')
axes[0, 0].set_ylabel('Amplitude')

#plotting the power spectrum for noisy signal
axes[0, 1].plot(freq3, power3)
axes[0, 1].set_title('Power Spectrum of Noisy Signal')
axes[0, 1].set_xlabel('Frequency (Hz)')
axes[0, 1].set_ylabel('Power')

#plot the cleaned signal and compare to original signal
axes[1, 0].plot(i, y_cleaned, label='Recovered Signal')
axes[1, 0].plot(i, x_i, label='Original Signal')
axes[1, 0].set_title('Recovered Signal without Noise, and original Signal')
axes[1, 0].set_xlabel('i')
axes[1, 0].set_ylabel('Amplitude')
axes[1, 0].legend()

#plot the power spectrum after removing noise
axes[1, 1].plot(freq3, power_cleaned)
axes[1, 1].set_title('Power Spectrum after Removing Noise')
axes[1, 1].set_xlabel('Frequency (Hz)')
axes[1, 1].set_ylabel('Power')

fig.tight_layout()
plt.show()

def detrend_signal(f_0, f_1, i):
  return 2 * np.sin(np.pi * f_0 * i) + np.cos(np.pi * f_1 * i)
```

```
i = np.linspace(0, 512, 512)
y_detrend = detrend_signal(f_0, f_1, i) + r_i

#run the FFT
fourier4 = np.fft.fft(y_detrend).real
freq4 = scipy.fft.fftfreq(len(i), d=i[1]-i[0])
power4 = np.abs(fourier4)**2

#ceate subplots
fig, axes = plt.subplots(ncols=2, figsize=(13, 4))

#plotting the detrended noisy signal
axes[0].plot(i, y_detrend)
axes[0].set_title('Detrended Noisy Signal')
axes[0].set_xlabel('i')
axes[0].set_ylabel('Amplitude')

#plotting the power spectrum
axes[1].plot(freq4, power4)
axes[1].set_title('Power Spectrum of Detrended Noisy Signal')
axes[1].set_xlabel('Frequency (Hz)')
axes[1].set_ylabel('Power')

plt.show()

#set all frequency components below 9 to zero
fourier4[np.where(power4 < 9)] = 0

#run the inverse FFT to recover the signal without noise
y_cleaned = np.fft.ifft(fourier4).real

#add the trend
y_filtered = y_cleaned + g(i)

#power spectrum after removing noise
power_cleaned = np.abs(fourier4)**2

#ceate subplots
fig, axes = plt.subplots(ncols=2, figsize=(13, 4))

#plot the cleaned signal and compare to original signal
axes[0].plot(i, y_filtered, label='Recovered Signal')
```

```
axes[0].plot(i, x_i, label='Original Signal')
axes[0].set_title('Recovered Signal and Original Signal')
axes[0].set_xlabel('i')
axes[0].set_ylabel('Amplitude')

#plot the power spectrum after removing noise
axes[1].plot(freq3, power_cleaned)
axes[1].set_title('Power Spectrum of Detrended Noisy Signal')
axes[1].set_xlabel('Frequency (Hz)')
axes[1].set_ylabel('Power')

plt.show()

#load data from file
data = np.loadtxt('phys581-binary.txt')
time = data[:,0]
count_rate = data[:,1]

#compute FFT
freq = np.fft.fftfreq(len(count_rate), time[1] - time[0])
fft = np.fft.fft(count_rate)
amp = abs(fft)

#ceate subplots
fig, axes = plt.subplots(ncols=2, figsize=(13, 4))

#plot the cleaned signal and compare to original signal
axes[0].plot(time, count_rate)
axes[0].set_title('Lightcurve of Binary System')
axes[0].set_xlabel('Time (s)')
axes[0].set_ylabel('Count-rate of Photons')

#plot the power spectrum
axes[1].plot(freq, amp)
axes[1].set_title('Amplitude Spectrum')
axes[1].set_xlabel('Frequency (Hz)')
axes[1].set_ylabel('Amplitude')

plt.show()

def check_power(n):
    return np.log2(n).is_integer()
```

```
#load data from file
heart_beats = np.loadtxt('phys581-beats.txt')

if check_power(len(heart_beats)):
    print("Length of data points is a power of two")
else:
    print("Length of data points is not a power of two")

#compute FFT
time_steps = 1/125
fft2 = np.fft.fft(heart_beats)
freq = np.fft.fftfreq(len(heart_beats), time_steps)
amplitude = np.abs(fft2)

#plot amplitude spectrum
plt.plot(freq, amplitude)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Aplitude Spectrum')
plt.show()

#load data from file
data = np.loadtxt('phys581-sunspots.txt')
months = data[:,0]
sunspots = data[:,1]

#plot
plt.plot(months, sunspots)
plt.xlabel('Time (Months)')
plt.ylabel('# of Sunspots')
plt.show()

#estimate length of the cycle
diff = np.diff(sunspots)
indx = np.where(diff > 0)[0]
cycles = np.diff(indx)
cycle = np.mean(cycles)
print("The length of the cycle is approximately", cycle, "months")

def check_power(n):
    return np.log2(n).is_integer()

if check_power(len(sunspots)):
    print("Length of data points is a power of two")
else:
    print("Length of data points is not a power of two")



def rungeKutta(function_list, initial, step, steps):

  # initialize lists
  x = []
  y = []
  z = []
  t = []

  # add initial values to lists
  for i, j in enumerate([t, x, y, z]):
    j.append(initial[i])


  for i in range(steps):

    k1 = []
    k2 = []
    k3 = []
    k4 = []

    for i in function_list:
      k1.append(step * i(x[-1], y[-1], z[-1]))

    for i, j in enumerate(function_list):
      k2.append(step * j(x[-1] + k1[i] / 2, y[-1] + k1[i] / 2, z[-1] + k1[i] / 2))

    for i, j in enumerate(function_list):
      k3.append(step * j(x[-1] + k2[i] / 2, y[-1] + k2[i] / 2, z[-1] + k2[i] / 2))

    for i, j in enumerate(function_list):
      k4.append(step * j(x[-1] + k3[i], y[-1] + k3[i], z[-1] + k3[i]))

    for i, j in enumerate([x, y, z]):
      j.append(j[-1] + (k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6)

    t.append(t[-1] + step)
  return t, x, y, z

def function_x(x, y, z, sigma = 10):
  return sigma * (y - x)

def function_y(x, y, z, r = 28):
  return r * x - y - x * z

def function_z(x, y, z, b = 8/3):
  return x * y - b * z

initial = (0, 5, 5, 5)

step = 0.01

steps = 2000

t_list, x_list, y_list, z_list = rungeKutta([function_x, function_y, function_z],
                                            initial,
                                            step,
                                            steps)

fig = plt.figure(figsize=(30,5))
ax = fig.add_subplot(111, projection='3d')
p = ax.scatter(x_list, y_list, z_list, c=t_list)

# Add color bar
cb = fig.colorbar(p)
cb.set_label("Time")

# Set labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Show plot
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
from PIL import Image
```

```
# Set up the figure and axes objects
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111, projection='3d')

# Define the update function for the animation
def update(frame):
    ax.clear()
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Frame {}'.format(frame))
    ax.plot(x_list[:frame], y_list[:frame], z_list[:frame], c='r', label="solution")
    ax.plot(np.sqrt(27), np.sqrt(27), 27/(8/3), marker='o',label='Fixed Point')
    ax.legend()

# Create the animation object
animation = FuncAnimation(fig, update, frames=600, interval=1)

# Save the animation as a .gif file using Pillow
animation.save('3d_plot_animation.gif', writer='pillow')

initial_conditions = [(0,5,5,5),(0,0,0,0), (0,0.1,0.1,0.1), (0,0,0,20),
                      (0,100,100,100), (0, 8.5, 8.5, 27)]

runge_kutta_list = []

for i in initial_conditions:
  runge_kutta_list.append(rungeKutta([function_x, function_y, function_z],
                                       i,
                                       step,
                                       steps))

fig = plt.figure(figsize=(15, 10))
for i in range(len(initial_conditions)):
  t_list, x_list, y_list, z_list = runge_kutta_list[i]
  ax = fig.add_subplot(2, 3, i+1, projection='3d')
  ax.plot(x_list, y_list, z_list, label="solution")
  ax.scatter([0, np.sqrt(27)],[0, np.sqrt(27)],[0, 27/(8/3)],
             label="Fixed Points", c='r')
  ax.scatter(x_list[-1], y_list[-1], z_list[-1], label="final point", c='y')
  ax.set_xlabel("X")
  ax.set_ylabel("Y")
  ax.set_zlabel("Z")
  ax.set_title("Initial Condition: " + str(initial_conditions[i]))
  ax.legend()


plt.show()

r = [14.6,14.75,60,90]
initial = [0, 5, 5, 5]

runge_kutta_list = []

for j in r:
  def function_y(x, y, z, r = j):
    return r * x - y - x * z

  runge_kutta_list.append(rungeKutta([function_x, function_y, function_z],
                                     initial,
                                     step,
                                     steps))




fig = plt.figure(figsize=(10, 10))

for i in range(len(runge_kutta_list)):
  t_list, x_list, y_list, z_list = runge_kutta_list[i]
  ax = fig.add_subplot(2, 2, i+1, projection='3d')
  ax.plot(x_list, y_list, z_list, label="solution")
  ax.scatter([0, np.sqrt(27)],[0, np.sqrt(27)],[0, 27/(8/3)],
             label="Fixed Points", c='r')
  ax.scatter(x_list[-1], y_list[-1], z_list[-1], label="final point", c='y')
  ax.set_xlabel("X")
  ax.set_ylabel("Y")
  ax.set_zlabel("Z")
  ax.set_title("R = " + str(r[i]))
  ax.legend()


plt.show()

r = np.linspace(0,30, 100)
```

```
initial = [0, 5, 5, 5]

runge_kutta_list = []

for j in r:
  def function_y(x, y, z, r = j):
    return r * x - y - x * z

  runge_kutta_list.append(rungeKutta([function_x, function_y, function_z],
                                     initial,
                                     step,
                                     steps))

x_list = []
for i in range(len(r)):
  x_list.append(runge_kutta_list[i][1][-1])

plt.scatter(r, x_list, s=0.5)
plt.xlabel("r value")
plt.ylabel("x value")

r = [1, 3, 13, 14.5, 26]
initial = [0, 5, 5, 5]

runge_kutta_list = []

for j in r:
  def function_y(x, y, z, r = j):
    return r * x - y - x * z

  runge_kutta_list.append(rungeKutta([function_x, function_y, function_z],
                                     initial,
                                     step,
                                     steps))

fig, axes = plt.subplots( len(runge_kutta_list), 3, figsize=(20,30))

for i in range(len(runge_kutta_list)):
  t_list, x_list, y_list, z_list = runge_kutta_list[i]
  freq = fftfreq(len(t_list), t_list[1]-t_list[0])
  args = np.argsort(freq)
  axes[i][0].plot(t_list, x_list)
  axes[i][0].set_xlabel("t")
  axes[i][0].set_ylabel("X")
  axes[i][1].plot(x_list, y_list)
  axes[i][1].set_xlabel("X")
  axes[i][1].set_ylabel("Y")
  axes[i][1].set_title("r = " + str(r[i]))
  axes[i][2].plot(freq[args], np.abs(fft(x_list))[args])
  axes[i][2].set_xlabel("frequency")
  axes[i][2].set_xlim([-7,7])
  axes[i][2].set_ylabel("amplitude")

plt.savefig("2_1_1.png")

rk4_212 = {}

for i, j in enumerate(r):
  rk4_212[j] = [runge_kutta_list[i][0], runge_kutta_list[i][1]]

for i in rk4_212.keys():
  rk4_212[i][0] = fftfreq(len(rk4_212[i][0]), rk4_212[i][0][1]-rk4_212[i][0][0])
  rk4_212[i][1] = fft(rk4_212[i][1])

r_data = []
freq = []
ampl = []
for i in rk4_212.keys():
  r_data.append(np.full(len(rk4_212[i][0]), i))
  freq.append(rk4_212[i][0])
  ampl.append(np.abs(rk4_212[i][1]))

print(len(r_data), len(freq), len(ampl))

plt.figure(figsize=(10, 6))

for i in range(len(r_data)):
  plt.scatter(r_data[i], freq[i], c = ampl[i], label = "r=" + str(r[i]), cmap='jet')

plt.xlabel("r value")
plt.ylabel("frequency")
cbar = plt.colorbar()
cbar.set_label("Amplitude")
plt.ylim(-4,4)
plt.show()

from google.colab import drive
drive.mount('/content/drive')

solar_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/phys581-sunspots.txt",
                         sep="\t", names=["Month", "Number Of Sunspots"])

value = 3143
while (np.sqrt(value) % 10 != 0):
  value +=1

print(solar_data)
print(value)

amp_solar = np.abs(fft(solar_data["Number Of Sunspots"].values, 3600))
freq_solar = fftfreq(len(amp_solar), 1)
args = np.argsort(freq_solar)
plt.plot(freq_solar[args], amp_solar[args])

power_solar = amp_solar**2
plt.plot(range(len(amp_solar)), (power_solar))
plt.xlabel("k")
plt.ylabel("$|c_k|^2$")

non_zero = [x for x in range(len(power_solar)) if power_solar[x] > 1e8]
print(non_zero)

plt.plot(solar_data["Month"], solar_data["Number Of Sunspots"].values * 100)
plt.xlabel("Months")
plt.ylabel("Number of Sunspots")

stock_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/phys581-stocks.txt", sep="\t")
print(stock_data)

fig = plt.figure(figsize=(20, 10))

for i in range(1,len(stock_data.columns)):
  ax = fig.add_subplot(2, 3, i)
  ax.plot(stock_data[stock_data.columns[i]])
  ax.set_xlabel("time unit")
  ax.set_ylabel("price")
  ax.set_title(str(stock_data.columns[i]))

R_list = []
for i in stock_data.columns[1:]:
  temp_data = stock_data[i]
  temp_r = [np.log(temp_data[i+1] / temp_data[i]) for i in range(len(temp_data)-1)]
  R_list.append(temp_r)

fig = plt.figure(figsize=(20, 10))
for i in range(len(R_list)):
  ax = fig.add_subplot(2, 3, i+1)
  ax.plot(range(len(R_list[i])), R_list[i])
  ax.set_xlabel("i")
  ax.set_ylabel("$R_i$")
  ax.set_title(stock_data.columns[1:][i])

correlate_orig = []

for i in stock_data.columns[1:]:
  temp_data = stock_data[i]
  mean = np.mean(temp_data)
  var = np.var(temp_data)
  ndata = temp_data - mean

  acorr = np.correlate(ndata, ndata, 'full')[len(ndata)-1:]
  acorr = acorr / var / len(ndata)
  correlate_orig.append(acorr)

fig = plt.figure(figsize=(20, 10))
for i in range(len(R_list)):
  ax = fig.add_subplot(2, 3, i+1)
  ax.plot(range(len(correlate_orig[i])), correlate_orig[i])
  ax.set_xlabel("k")
  ax.set_ylabel("Autocorrelation $C_k$")
  ax.set_title(stock_data.columns[1:][i])

power_stock = []
freq_stock = []

for i in stock_data.columns[1:]:
  temp_data = stock_data[i]
  temp_data = np.abs(fft(temp_data.values))**2
  temp_freq = fftfreq(len(temp_data), 1)
  power_stock.append(temp_data)
  freq_stock.append(temp_freq)

fig = plt.figure(figsize=(20, 10))
for i in range(len(R_list)):
  ax = fig.add_subplot(2, 3, i+1)
  args = np.argsort(freq_stock[i])
  ax.plot(freq_stock[i][args], power_stock[i][args])
  ax.set_xlabel("frequency")
  ax.set_ylabel("Power Spectrum")
  ax.set_title(stock_data.columns[1:][i])

dow_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/phys581-dow.txt",
                       header=None)

power_dow = np.abs(fft(dow_data[0].values))**2
freq_dow = fftfreq(len(power_dow), 1)
args = np.argsort(freq_dow)

fig, axes = plt.subplots( 1, 2, figsize=(10,3))

axes[0].plot(dow_data[0])
axes[0].set_xlabel("time unit")
axes[0].set_ylabel("price")
axes[1].plot(freq_dow[args], power_dow[args])
axes[1].set_xlabel("frequency")
axes[1].set_ylabel("Amplitude")

temp_data = dow_data[0]
mean = np.mean(temp_data)
var = np.var(temp_data)
ndata = temp_data - mean

acorr = np.correlate(ndata, ndata, 'full')[len(ndata)-1:]
acorr = acorr / var / len(ndata)

plt.plot(range(len(acorr)), acorr)
plt.xlabel("k")
plt.ylabel("Autocorrelation $C_k$")

co_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/phys581-co2.txt",sep='\t')
```

```
series = []
for i in range(co_data.shape[0]):
  series.extend(co_data.iloc[i][1:-1])

plt.plot([x for x in series if x >= 0])

fig, axes = plt.subplots( 1, 2, figsize=(10,3))

axes[0].plot(co_data.iloc[-1][1:-1])
axes[0].set_xlabel("Months")
axes[0].set_ylabel("$CO_2$ Concentration")
axes[1].plot(co_data['Year'], co_data['Jul'])
axes[1].set_xlabel("Year")
axes[1].set_ylabel("$CO_2$ Concentration")

linear_fit = np.polyfit(co_data["Year"].values, co_data["Jul"].values, 1)

def linear_function(t, a= linear_fit[0], b= linear_fit[1]):
  return a * t + b


co_normalized = co_data.copy(deep=True)
for i in co_normalized.columns[1:-1]:
  for j, k in zip(range(co_normalized.shape[0]), co_normalized["Year"].values):
    co_normalized.at[j, i] = co_normalized.at[j, i] - linear_function(k)

series = []
for i in range(co_normalized.shape[0]):
  series.extend(co_normalized.iloc[i][1:-1])

series = [x for x in series if x >= 0]
plt.plot(series)
plt.xlabel("Months")
plt.ylabel("$CO_2$ Concentration")

co_amplitude = np.abs(fft(series))
co_frequency = fftfreq(len(series), 1)
args = np.argsort(co_frequency)

plt.plot(co_frequency[args], co_amplitude[args])
plt.xlabel("frequency")
plt.ylabel("")
```

```
power_co = co_amplitude**2
plt.plot(co_frequency[args], power_co[args])
plt.xlabel("frequency")
plt.ylabel("Power Spectrum")

t = [2000]
toxic_values = [linear_function(t[-1])]
while toxic_values[-1] < 7e4:
  t.append(t[-1] + 100)
  toxic_values.append(linear_function(t[-1]))

plt.plot(t, toxic_values, label="Trend")
plt.plot(t, np.full(len(t), 7e4), label="Toxicity Treshold")
plt.xlabel("Year")
plt.ylabel("$CO_2$ Concentration")
plt.legend()
```

References

[1] Yi-Chao Xie et.al Flow Topology Transition via Global Bifurcation in Thermally Driven Turbulence https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.120.214501