PHYS381
Python Data Analysis

Lily Williams
ID: 42415299
lfw25@uclive.ac.nz

PHYS381 22S2

July 2022

1 Statistics

Data acquisition is never a perfect act, and any good scientist knows that measurements should be repeated to ensure accuracy. When measurements are repeated, however, it is more than likely that the data will vary slightly for a number of reasons. In order to then make valid conclusions from a varying data-set, it is important to use statistical tools to understand the associated patterns. These statistical tools can be used to predict the behaviour of a system over long periods of time, and can reinforce the strength of any results obtained.

A common distribution of data is the Gaussian distribution, often referred to as simply "a Gaussian", which observes a bell-shape. The Gaussian has a probability density function which is represented mathematically by Equation 1,

$$P(\gamma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(\gamma - \overline{\gamma})^2}{2\sigma^2}\right] \tag{1}$$

where γ is the mean of the data and σ is the standard deviation. Another important quantity is the variance, σ^2, which is the square of the standard deviation. Comparing these characteristic values between two data-sets can reveal much about their relationship. A measure of how significant these differences are can be represented by a probability that follows the formula described in 2,

$$P = \beta\left(\frac{df}{df + t^2}, \frac{df}{2}, \frac{1}{2}\right) \tag{2}$$

where β is the incomplete beta function, df is the number of degrees of freedom in the data-set, and t is a conventional measure of statistical significance. When the probability P is small – less than 0.01 – this means that the result is statistically significant; a P of 0.01, or 1%, says that the probability of the means being different is 99%. The *SciPy* statistics library for Python data analysis, *scipy.stats*, has a function to calculate the significance of two data-sets, *ttest_ind*. However, the P-value returned by this particular significance test is only valid if the data-sets have the same variance. Determining whether the variances are the same can be done either directly or with an "f-test", which is a particular type of statistical test useful in linear regression models. If the P-value produced by the f-test is less than the significance, the data-set provides enough evidence to conclude that the linear regression model fits the data better than a model with no independent variables.

Exercise 1

The first exercise consisted of reading in two data files associated with two unknown samples and determining if their means are different to any notable significance. The first data file, *SampleA.dat*, contained 100 rows of floats. The second data file, *SampleB.dat*, contained 125 rows of floats.

Firstly, the raw data was plotted on a histogram using *MatPlotLib* to see the general distribution. These histograms are shown in Figure 1[1]. From the plot, it was clear that the data followed a Gaussian distribution, so, the mean, variance, and standard deviation of each of the samples was determined using the *NumPy* library. These results are listed in Table 1.

[1]The sample data was provided with no context, so no axes labels are used. It is assumed that the data is arbitrary beyond the counts in each bin.

Table 1: This table summarises the Gaussian characteristics of Samples A and B.

Data-set	**Mean γ**	**Standard Deviation σ**	**Variance σ^2**
Sample A	-0.3579	1.941	3.767
Sample B	-0.2501	1.903	3.621

Using this data, the Gaussian distribution for each sample set was overlaid on the histograms for a direct visual comparison, shown in Figure 1.

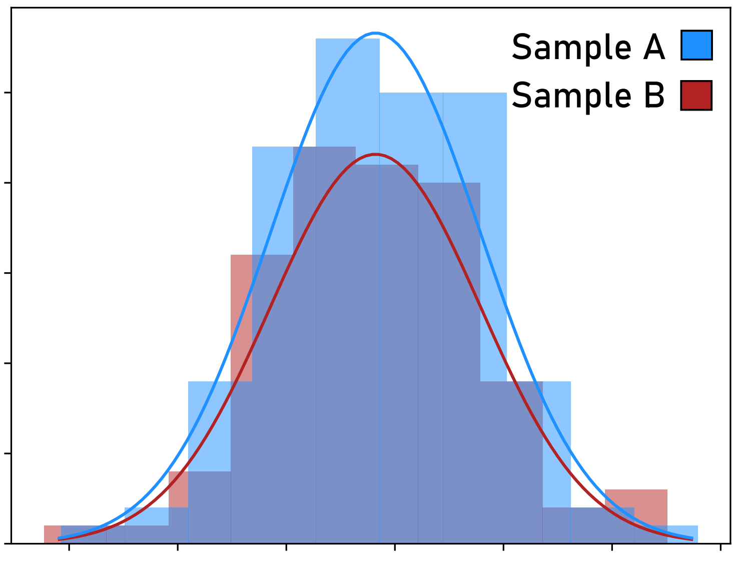


Figure 1: A visual representation of the two data-sets. The histogram is a direct plot of the data, with arbitrary x and y axes, and the Gaussian overlay is plotted from the sample distribution properties mentioned in Table 1.

From a direct visual inspection of this graph, the Gaussian distributions were observed to be extremely similar – barring amplitude. This, and the agreement in the variances to the first significant figure, was enough to suggest that the t Test would be appropriate for this data-set. Therefore, the *scipy.stats ttest_ind* function was used to calculate the significance. This significance was $P = 0.6788$ (to 4 s.f.), a very high value. Normally, one would look for a P value of less than $P = 0.05$, or 5%, to prove the mean is significantly different. Since this value is so high, it indicates that the means of the two data-sets are not significantly different, or, put simply, the two means are the same.

Exercise 2

In a similar vein to Exercise 1, Exercise 2 was to read in two very large data-sets containing climate change data. The same as previously, the purpose was to determine if the means were the same and to what significance level. The first data file, *anthropogenic.dat*, contained 1,140,113 data points describing variation in temperature anomalies from the mean assuming that humans are responsible for climate change, and the second data file *natural.dat*, contained 3,214,097 data points assuming climate change derives from natural sources, such as variations in solar insolation. Reading in this data, it also had to be cleaned of invalid data formats, like NaN lines. This was done using the *Pandas* library's *read_csv* and *dropna* functions to read the data into DataFrames and clean it up, and then it was converted back into *Numpy* arrays to speed up any calculations.

With the same process as in the first exercise, the data was plotted on a histogram and the Gaussian characteristics were determined using python data analysis packages. These characteristics, listed in Table 2, were used to overlay the Gaussian curves on the histograms. This is shown in Figure 2.

Table 2: This table summarises the Gaussian characteristics of the Anthropogenically caused and Naturally caused climate change-related temperature anomalies.

Data-set	**Mean γ**	**Standard Deviation σ**	**Variance σ^2**
Anthropogenic	0.7496	1.832	3.356
Natural	-0.2660	1.824	3.327

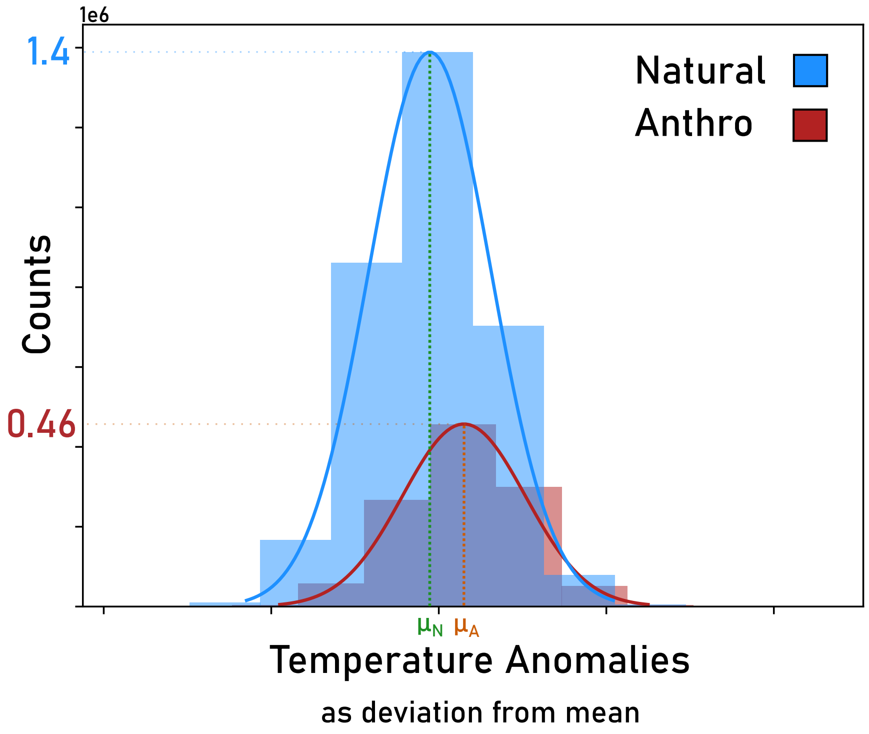


Figure 2: A visual representation of the two climate change data-sets with the Gaussian curves overlaid.

Clearly, the variances are very similar, even more similar than those in the first exercise. For this reason, the *scipy ttest* was again used to determine the significance of the means being different. This P value was determined to be $P = 0$, suggesting there is a 0% chance of the means being the same. This initially seems like an error, especially considering Figure 2 depicts the Anthropogenic Gaussian fully within the Natural Gaussian, but when the Gaussians are normalised, as is shown in Figure 3, it can be seen that these distributions are notably distinct.

When the distributions are considered alongside the *vast* sample size, it becomes a near certainty that any pair of randomly selected data points, one from each set, will be quite distinct. This is different than the situation in the first exercise where the data-sets were significantly smaller. Therefore, there is a near guarantee that the means of the two data-sets in this exercise are different. In the context of this exercise, it means that if the real data reflecting climate anomalies were compared to these data-sets, it would be

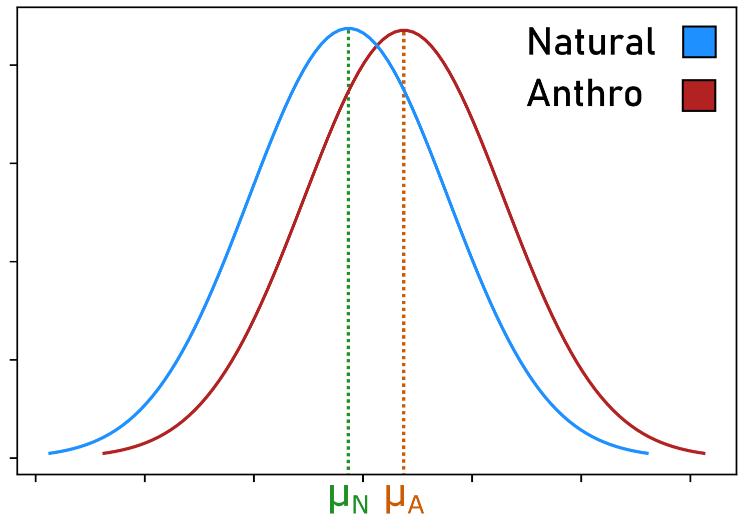


Figure 3: A normalised Gaussian distribution of both the Anthropogenic and Natural data.

possible to determine the most likely cause of climate change without having the complication of indistinct comparators.

2 Correlation

When measuring variables in an experiment, it is sometimes of interest to determine if there is a direct relationship between two variables. Correlation between variables can demonstrate a dependence, or be used to show that there is no relationship. Uncorrelated variables are called "independent", which is a key concept in relationship analysis. The most common, and most easily determined, correlation is a linear relationship. If you have two sets of data x and y, if x is plotted against y, most often as a scatter plot, and the line of best fit is linear, it can be said that there is a linear correlation, which can be quantified by a linear correlation coefficient r. This coefficient r can be calculated using Equation 3, or more efficiently using statistics tools provided in a number of python packages.

$$r = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \overline{y})^2}} \tag{3}$$

When there is the maximum positive linear correlation between two data-sets, that is, an increase in x correlates with an equal increase in y, the value of r will be 1. When there is maximum negative correlation, when an increase in x correlates with an equal decrease in y, r will be -1. A value of $r = 0$ indicates no correlation between the two variables. The correlation coefficient does not contain any measure of statistical significance, however, because it does not contain any information about the distributions of the x or y data-sets used to calculate it. However, the significance can be computed using the same P-value calculation in Equation 2, or with a python package tool.

Exercise 3

Exercise 3 comprised of reading in a data file consisting of 3 columns, each with 99 data points. The first, second, and third columns were the data points for variables "X", "Y", and "Z", respectively. The task was to determine the correlation coefficient of X to Y and X to Z, take one-to-one tuples of (X_i, Y_i) and (X_i, Z_i) as coordinate data points, as well as to identify whether the statistical significance is at the 95% level ($P = 0.05$) or better. The X, Y, and Z data is contextless, and so the plot displaying the data does not have labelled axes.

The most informative graph for this kind of analysis is a scatter plot, so the data was cleaned using the *Pandas dropna* tool and then the two comparisons were plotted on the same graph, Figure 4.

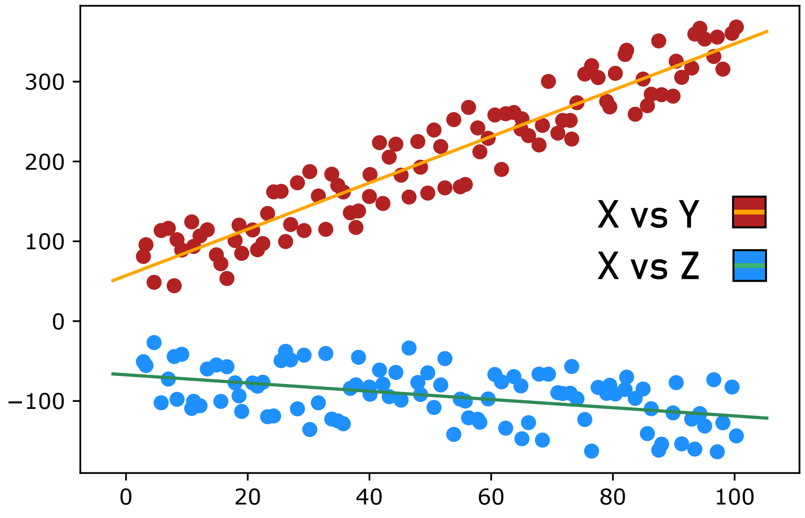


Figure 4: A scatter plot representing X vs Y in red, with the linear relationship represented in orange, and X vs Z in blue, with the relationship represented in green.

Using the *scipy.stats* library, the *linregress* linear regression tool calculated the correlation coefficient, statistical significance, and the standard error, as well as the slope m and intercept c of the line that best fits the data for both X vs Y and X vs Z. These values are all detailed in Table 3.

Table 3: This table summarises the linear regression characteristics of the X vs Y and X vs Z relationships.

Data-set	**Correlation r**	**P-value**	**Standard Deviation σ**	**Slope m**	**Intercept c**
X vs Y	0.9464	2.092×10^{-49}	0.1006	2.903	57.03
X vs Z	-0.4477...	3.373×10^{-6}	0.1047	-0.5152	-67.21

For the X vs Y comparison, the correlation coefficient was $r = 0.9464$, which is almost the maximum correlation. This indicates that there is a very strong positive correlation between X and Y. This relationship also has a significance of $P = 2.092 \times 10^{-49}$, which is extremely small. This means that this an extremely significant relationship.

For the X vs Z comparison, the correlation coefficient was $r = -0.4477$, a moderately negative correlation. This would indicate that X and Z are somewhat proportional, although there are likely other factors at play. This relationship has a significance of $P = 3.373 \times 10^{-6}$, which, while higher than the X vs Y significance, is still very small. Therefore, this is certainly a statistically significant relationship.

3 Fitting Linear Models

Often gathered data will appear to follow certain curves, and it can be useful to fit that data in order to both predict future and discover past behaviours. A fit is considered linear if it can be represented as a linear combination of polynomial basis functions, e.g. x, x^2, x^3 etc, or represented mathematically as

$$y(x) = \sum_{k=1}^{M} a_k X_k(x) \tag{4}$$

In the case of a straight line, the basis functions are 1 and x, but more complicated data-sets will often fit to higher order polynomials. The task of fitting a model is to find the fitting coefficients a_k for each of the polynomial factors which best match the data points. The best way to ensure that the fit is the most effective one is to minimise the error between the model and the data. This is called a "least-squares fit", and the measure of accuracy is called the "chi-squared" (χ^2). The χ^2 is effectively the sum of the difference between the fit and the data at each point along the curve, divided by the standard deviation σ, and then squared. This calculation is represented in Equation 5, where M is the number of parameters and N is the number of data points.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k X_k(x)}{\sigma_i} \right)^2 \tag{5}$$

When solving this problem by hand, it is actually solving a set of M simultaneous equations, however, there are a number of Python tools which will determine the fit coefficients a_k and its accuracy χ^2. The error in the accuracy is represented in the *covariance matrix*, which is also produced by these Python tools. The expected value of χ^2 after minimisation is $<\chi^2>= N - M$, and the variance of this expected value is $\sigma^2_{<\chi^2>} = 2(N - M)$. If χ^2 differs from $<\chi^2>$ by several $\sigma^2_{<\chi^2>}$, then it suggests that the model is incorrect.

Exercise 4

Exercise 4 consisted of creating 100 data points that vary from a predefined cubic polynomial by a random amount. Once these data points were generated, a model had to be fit to the data; this model should ideally be as close as possible to the original, predefined curve.

The data was generated and plotted alongside the predefined curve. Using the *Numpy polyfit* tool, the data points were fit to a polynomial of degree 3. This fit curve was also plotted on top of the predefined curve to visually see the accuracy of the fit. This graph is shown in Figure 5.

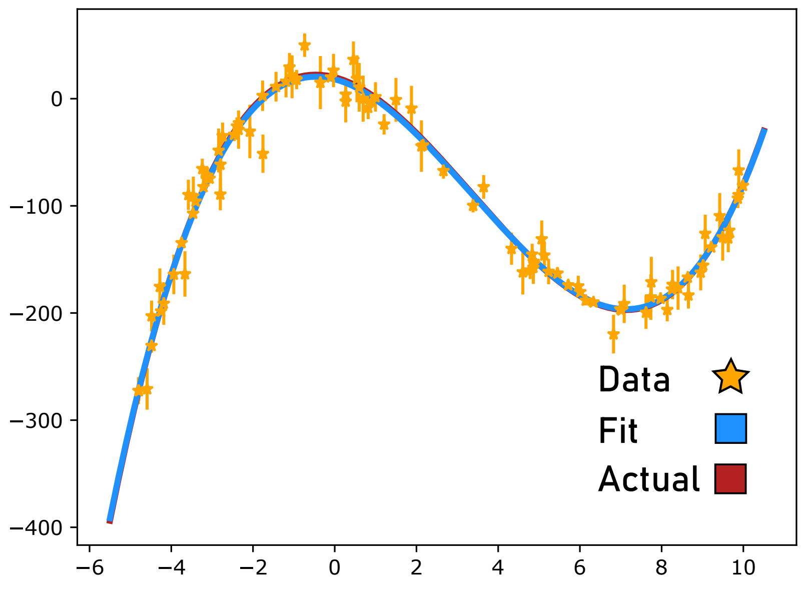


Figure 5: A scatter plot representing the predefined curve in red, the randomly varying data points in orange, and the *Numpy*-generated fit curve in blue.

Clearly the fit is very close to the predefined curve, overlaying almost exactly. The original and fit coefficients are shown in Table 4.

Table 4: This table summarises the coefficients of the real and model polynomial curves.

Curve	x^3	x^2	x	$+c$
Actual	1.000	-10.00	-10.00	20.00
Fit	1.032	-10.14	-11.22	19.58

The *scipy* statistics library has a *chisquare* tool which takes data points along the two curves and determines the χ^2 and significance of the fit. When data points along the curves listed in Table 4 were generated using the *Numpy polyval* function and passed into the aforementioned *chisquare* tool, the χ^2 value was $\chi^2 = -235.2$ and $P = 1.0$. The P value being 100% indicates that the difference between the two curves is absolutely minimal. In order to evaluate the efficacy of the model in relation to the χ^2,

$<\chi^2>$ and $\sigma^2_{<\chi^2>}$ were calculated as follows:

$$\begin{aligned} <\chi^2> &= N - M \\ &= 100 - 4 \\ &= 96 \\ \sigma^2_{<\chi^2>} &= 2(N - M) \\ &= 2(96) \\ &= 192 \\ \chi^2 &= <\chi^2> + \sigma^2_{<\chi^2>} x \\ -235.2 &= 96 + 192x \\ x &= -1.725 \end{aligned}$$

Since χ^2 varies by only $1.725\sigma^2_{<\chi^2>}$ from $<\chi^2>$, it is a reasonable conclusion that the model fits the data with a good level of accuracy.

4 Fitting Non-Linear Models

Oftentimes, data is not linear, and thus it is not such simple task of finding simple coefficients to basis parameters. Instead, in many cases, data may be best modeled by a non-linear function, such as a logarithm, exponential, or sinusoidal function, among others. Most simply, there are convenient Python packages that will accomplish these non-linear model fits, although the fit can be done using specific algorithms. In the case of fitting a sinusoidal model of regularly fluctuating temperature, of the kind explored in Exercise 5, the parameters might follow the formula laid out in Equation 6:

$$T(t) = A\cos(2\pi f t - \phi) + Bt + C \tag{6}$$

where T is the temperature as a function of time, t is the time in months, A is the amplitude of the cycle, f is the frequency, ϕ is the phase shift related to the hemispheres, B is the temperature gradient, and C is a constant. However, it is possible that multiple different models could be fit to the same data. Where Equation 6 contains a gradient slope term, a viable model may not have that term, such as Equation 7:

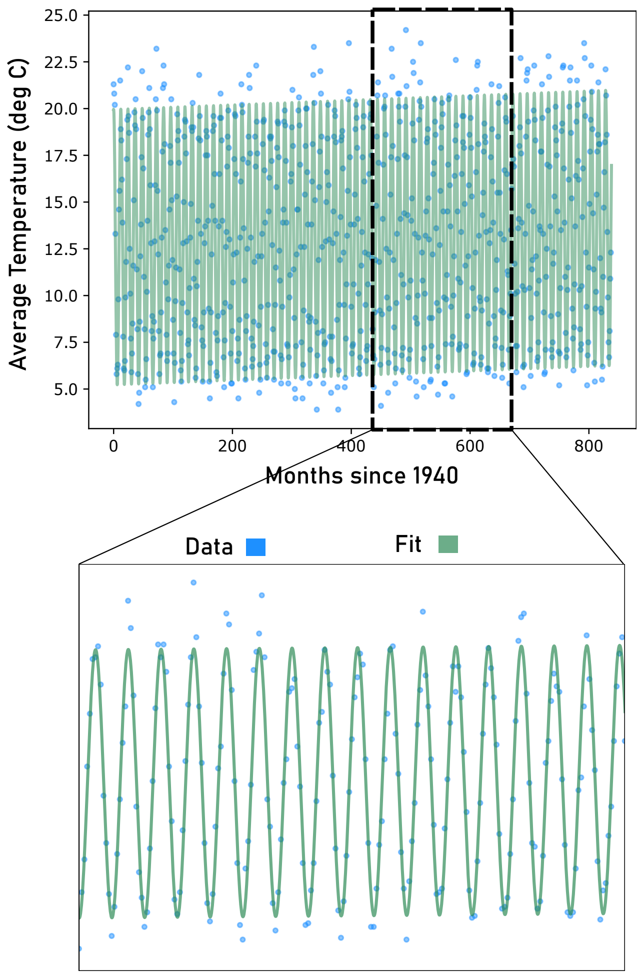
$$T(t) = A\cos(2\pi f t - \phi) + C \tag{7}$$

The accuracy of the model can be quantified by the values of the covariance matrix, which reflect the uncertainty in the parameters of the fit. The values in this matrix will vary based on many things. In the case of the *scipy.optimize.curve_fit* tool package, which is the one used in Exercise 5, the covariance matrix will vary based on initial guess parameters and the presumed data model. These uncertainties are not random, however, and will tend towards the same values with the same initial guess parameters. Many tool packages will produce the covariance matrix alongside the model parameters.

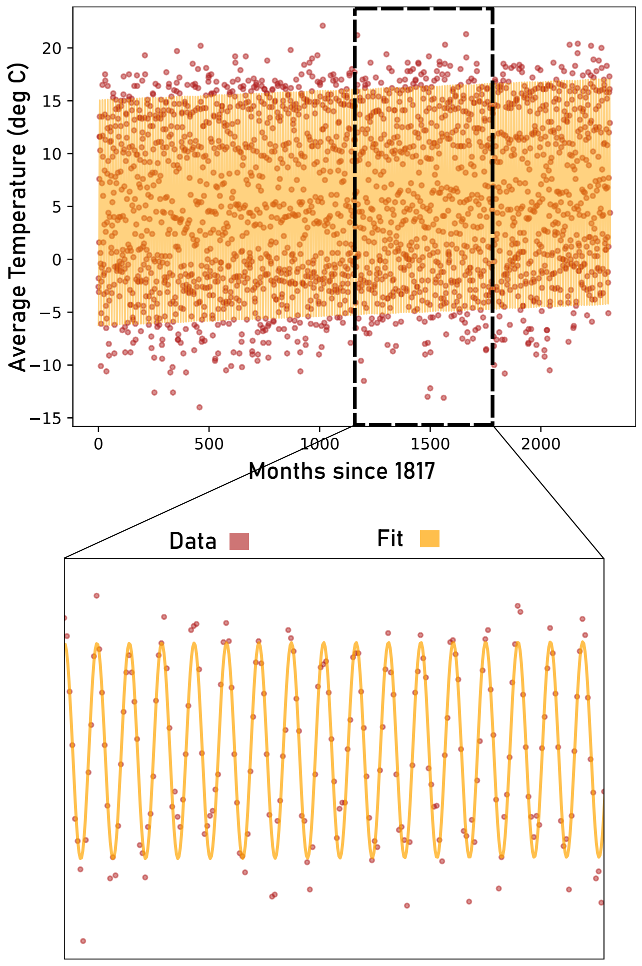
Exercise 5

Exercise 5 involved reading in data describing the average temperature per month in two cities: Canberra, which had 838 months of data from 1940 to 2010, and Oslo, which had 2314 months of data from 1817 to 2010. The task was to fit these data-sets to a sinusoidal model and to hence determine the sinusoidal parameters.

Two model fits were done using the *scipy.optimize.curve_fit* toolbox, one fitting to the model in Equation 6 and one to Equation 7. The fit to Equation 6 produced the amplitude A, frequency f, phase ϕ, temperature gradient B, and offset C. These parameters were used to plot the model curve on top of the actual data in order to visually determine the accuracy of the model, and is shown in Figure 6, where Figure 6a is the data for Canberra and Figure 6b is the data for Oslo. The parameters are also summarised in Table 5.



(a) Canberra



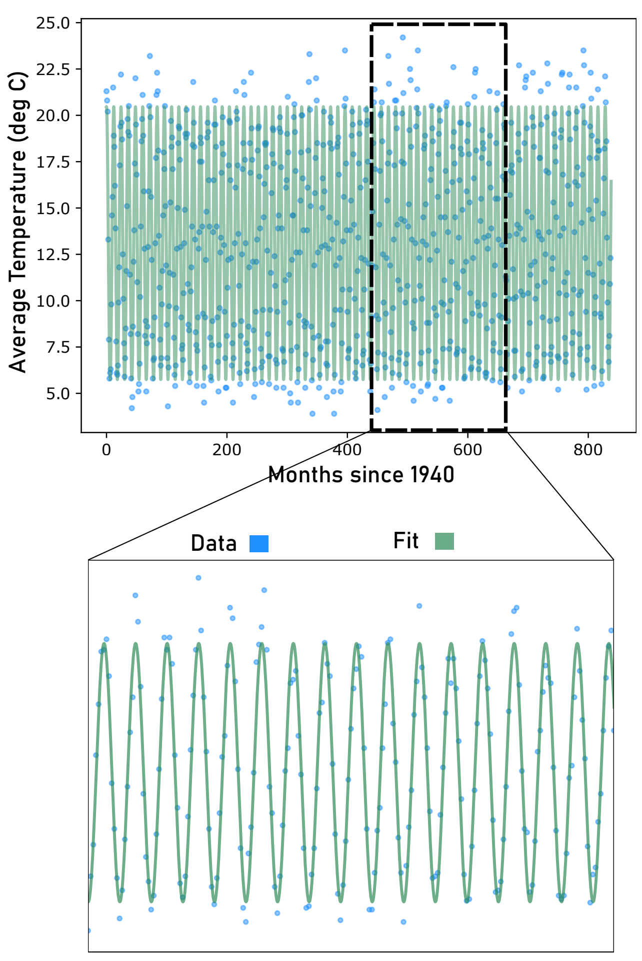
(b) Oslo

Figure 6: It can be seen here that the sinusoidal models accounting for a gradient slope, with parameters listed in Table 5, accurately fit the data for both cities.

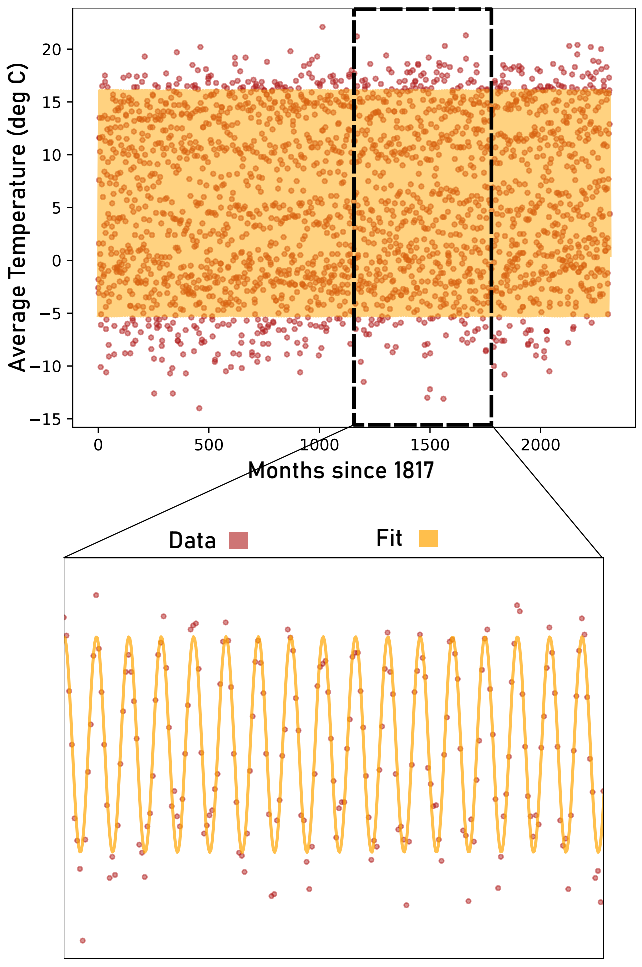
Table 5: This table summarises the coefficients of the Oslo and Canberra model polynomial curves when the model includes a gradient slope term.

Curve	A	f	ϕ	B	C
Canberra	7.364	0.08333	1.532	0.001235	12.59
Oslo	-10.68	0.08334	1.489	0.0008707	5.413

Alternatively, the data was also fit without the temperature gradient term, B, matching Equation 7. As with the sloped fit, the model was plotted atop the data to visually determine if the fit is accurate. These plots are shown in Figure 7 and the parameters are summarised in Table 6.



(a) Canberra



(b) Oslo

Figure 7: It can be seen here that the sinusoidal models that do not include a gradient slope, with parameters listed in Table 6, also accurately fit the data for both cities.

Table 6: This table summarises the coefficients of the Oslo and Canberra model polynomial curves when the model does not include a gradient slope term.

Curve	A	f	ϕ	C
Canberra	7.364	0.08333	1.532	13.10
Oslo	-10.68	0.08334	1.489	5.413

Clearly, both of these models fit the data fairly well, and, aside from the offset, have nearly identical parameters. A good way to determine the accuracy of the fits is to investigate the values in the covariance matrix, in particular, a good metric is taking the maximum value of the covariance matrix. These values for each data-set are summarised in Table 7.

Table 7: This table summarises the maximum covariance matrix values from each data-set and each model fit.

Curve	**Sloped**	**Flat**
Canberra	0.007057	0.003744
Oslo	0.007234	0.003910

The maximum covariance values for the sloped model is just under double the maximum covariance of the unsloped model. This could indicate that the model which includes the temperature gradient is more uncertain than the one which doesn't. While they both clearly fit and, objectively speaking, the uncertainty values are still quite small, this could be interpreted to mean that the temperature is not actually increasing over time; this would have some major implications for the conversation around climate change.

5 Fourier Analysis

Fourier Transforms are a powerful mathematical and analytical tool. In data analysis, particularly signal processing and other frequency-related data-sets, they can be used to take a muddled, complex sum of signals and filter out the individual frequencies. This is useful when trying to identify periodic signals or patterns within a data-set.

Fundamentally, the mathematics of a Fourier Transform involves transforming signals that exist in the spatial or temporal domains into frequency domains. Once in this frequency domain, the signals can be disentangled, and then an inverse fourier transform can return the signals from the frequency domain into the spatio-temporal domain.

In real laboratory data gathering, data is gathered at steady time intervals rather than continuously. This "discretely sampled" data can also be Fourier transformed by choosing a "sampling frequency" for the transform. There is, however, a lower limit to the sampling frequency, called the "Nyquist frequency", and is determined to by Equation 8.

$$f_N = \frac{1}{2\Delta t} = \frac{f_S}{2} \tag{8}$$

Another way of explaining the Nyquist frequency is the maximum signal that can be detected is equal to half the sampling rate. When the sampling frequency is lower than the Nyquist frequency, incorrect signals can be matched to discrete data points. An example of this is displayed in Figure 8, where the red line displays the real signal, where the sampling frequency has detected the marked data points, and the dashed blue line displays another "aliased" signal that has been fit to the detected data.

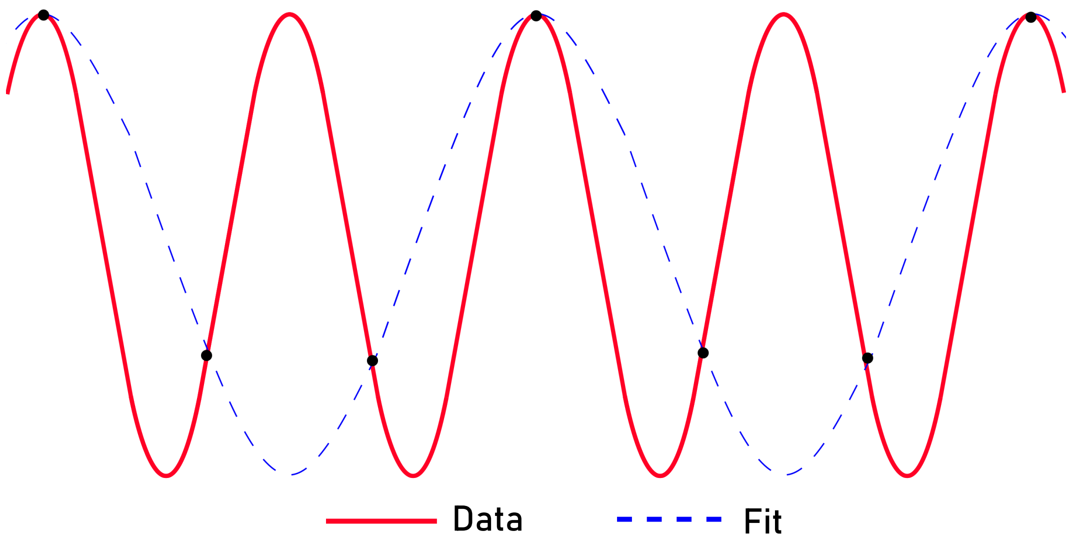


Figure 8: A visual representation of the effect of the Nyquist theorem (Heath).

Clearly, while the aliased signal fits the sampled data, it does not match the actual signal. It can therefore be seen that the sampling frequency must be higher than this Nyquist frequency, otherwise fits will not be accurate.

Exercise 6

Exercise 6 involved reading in some artificially created data and analysing the spectrum of an aliased signal. The data was created with the combination of a 5 Hz and a 35 Hz signal. When this discrete data is analysed with a sampling frequency of 1000 Hz – well above the Nyquist frequency – the signals are correctly identified. This is shown in Figure 9.

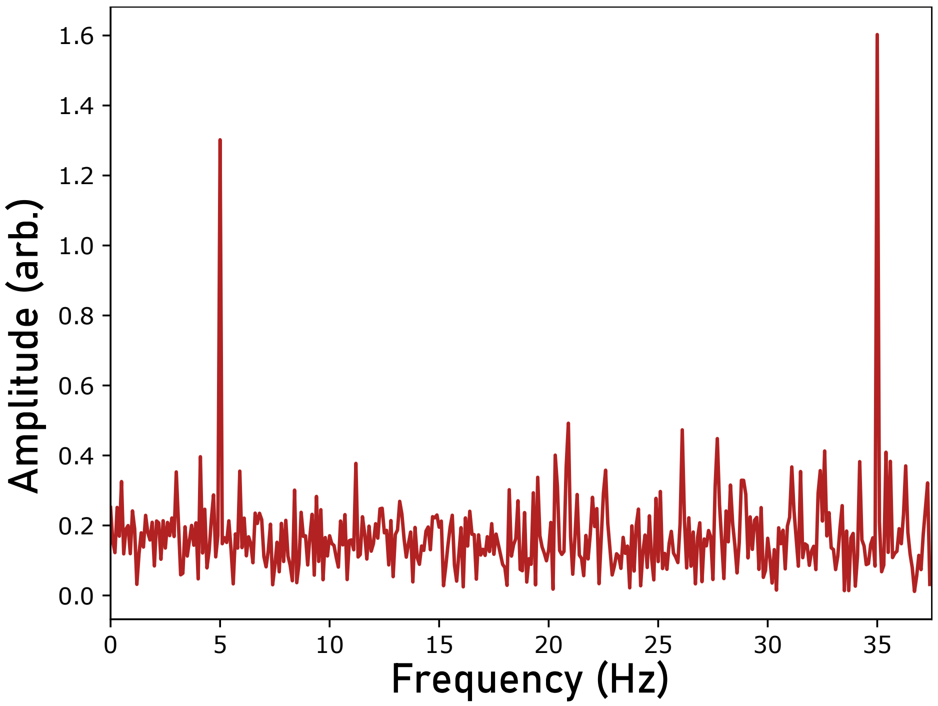


Figure 9: The individual signals are correctly identified when the sampling frequency is very high.

In order to analyse the effect of the sampling frequency on the aliasing of the signals, the 35 Hz signal was focused on, and a sweep was done from 1 Hz to 140 Hz. The guessed position of the 35 Hz signal was plotted against the sampling frequency in order to determine the relationship. This is shown in Figure 10.

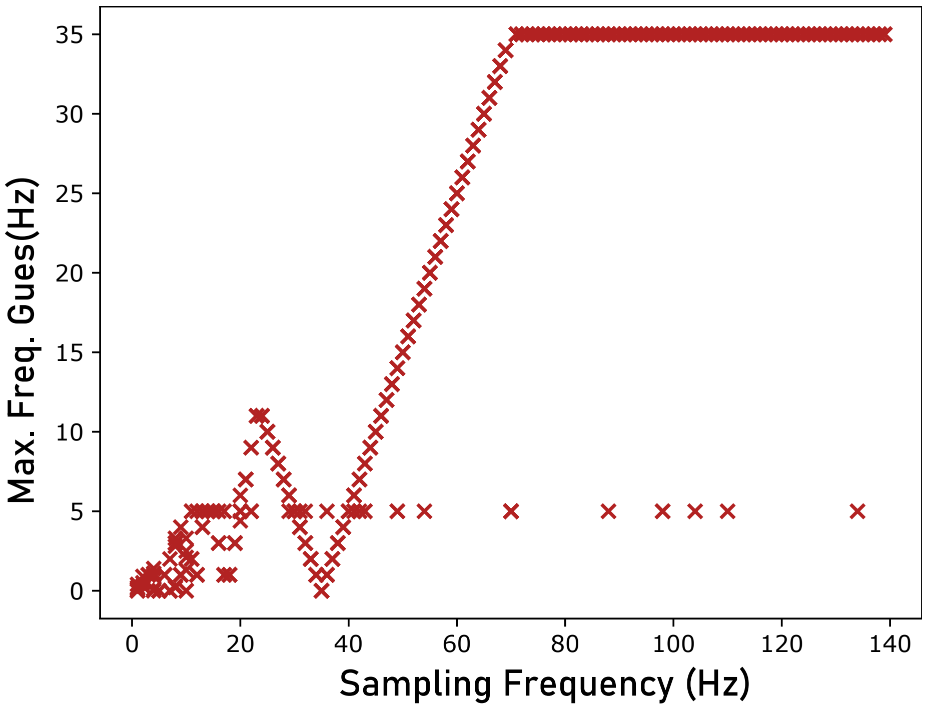


Figure 10: The relationship of the sampling frequency to the guessed position of the 35 Hz signal indicated the position of the Nyquist Frequency.

It can be seen from this figure that between 1 Hz and 35 Hz, the guess is quite random. From 35 Hz to 70 Hz, it steadily increases as the sampling frequency approaches the Nyquist frequency, where it flattens out into a flat line; these incorrect guesses are where the signal is aliased. This flat line, unsurprisingly, begins at the Nyquist frequency, where the 35 Hz signal is able to be reliably guessed. The seemingly random deviations from the straight line above 70 Hz correspond to multiples of 35, where the code used to calculate the signal cuts off the top data point.

Exercise 7

Exercise 7 included reading in real atmospheric tide data and using Fourier Transforms to determine the dominance of diurnal or semi-diurnal tides throughout the 24-hour cycle over 10 days. The task was to recreate a provided graph; the data manipulations had to be recreated from the final result. The first step taken was to read in the data; this produced two arrays: time, in days, and horizontal velocity of wind, in kilometres per hour. This raw data was plotted, and then passed through a Fourier transform to move the velocity data to the frequency domain. Given that the data was over 10 days at 24 hours a day, the sampling frequency was set to 240 Hz. Therefore, this sampling frequency is high enough to avoid signal aliasing. This data was then plotted on a logarithmic scale to match the supplied graph, and is show in Figure 11.

The semi-diurnal and diurnal signals are clearly visible by the red and blue lines, respectively, but it can also be seen that the red diurnal tide is dominant during this particular 10-day period.

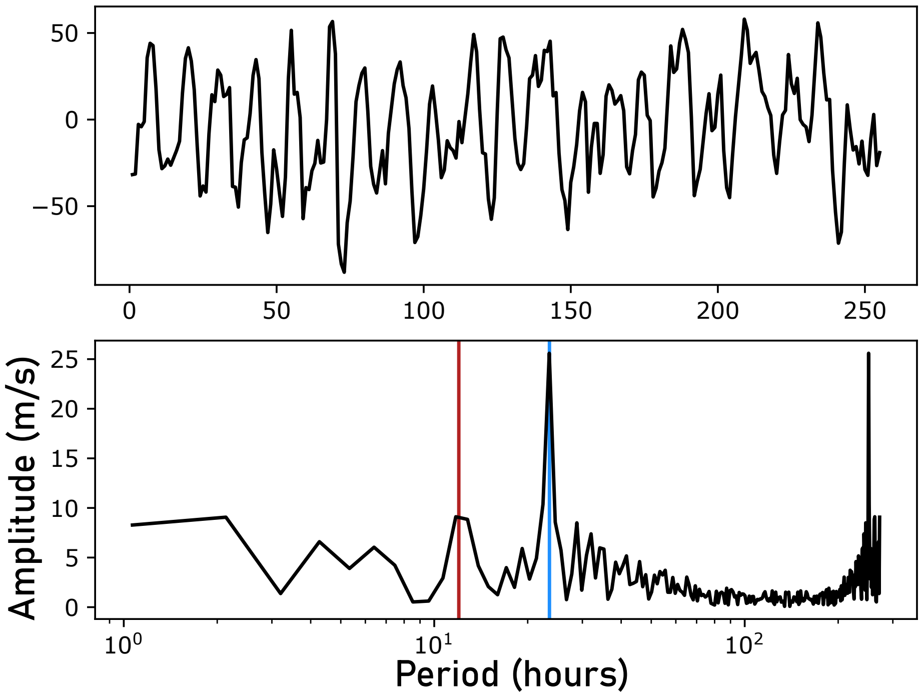


Figure 11: The raw data and the Fourier analysed data look significantly different, although the source frequencies have clearly been identified.

References

[1] Math.net *Gaussian Distribution.* https://www.math.net/gaussian-distribution

[2] Amitava Dasgupta, Amer Wahed *Gaussian Distribution.* https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gaussian-distribution

[3] Corporate Finance Institute *Correlation.* https://corporatefinanceinstitute.com/resources/knowledge/finance/correla

[4] LibreText *Fitting Linear Models to Data.* https://math.libretexts.org/Bookshelves/Algebra/Map%3A_College_Algeb

[5] OpenTextBC *Fitting Linear Models to Data.* https://opentextbc.ca/algebratrigonometryopenstax/chapter/fitting-linear-models-to-data/

[6] Allen Mclntosh *Fitting Linear Models.* SpringerNature, 1982, Volume 10

[7] Neil Polhemus *Fitting Nonlinear Regression Models.* https://www.statgraphics.com/blog/nonlinear_regression

[8] TheMulQuaBio *Model Fitting using Non-linear Least-squares.* https://mhasoba.github.io/TheMulQuaBio/notebooks ModelFitting-NLLS.html

[9] John M. Chambers *Fitting nonlinear models: numerical techniques.* https://www.jstor.org/stable/2334900

[10] Janet Heath *Nyquist Theorem.* https://www.analogictips.com/adcs-sufficient-sampling-nyquists-rate/

[11] R. Picard *Fourier Analysis.* International Encyclopedia of the Social Behavioral Sciences, Pergamon, 2001, Pages 5754-5760

[12] David Morin *Fourier analysi.* https://scholar.harvard.edu/files/david-morin/files/waves$_f$*ourier.pdf*

Appendix

Appendix - Exercise Code

```
### Ex 1

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stat



sampleA = np.genfromtxt("Files/SampleA.dat", delimiter=',',skip_header=1) # Read in data A
sampleB = np.genfromtxt("Files/SampleB.dat", delimiter=',',skip_header=1) # Read in data B

sampleAMean = np.mean(sampleA)
sampleBMean = np.mean(sampleB)

sampleAVar = np.var(sampleA)
sampleBVar = np.var(sampleB)

sampleAStd = np.std(sampleA)
sampleBStd = np.std(sampleB)

pVal = stat.ttest_ind(sampleA, sampleB).pvalue
fTest = stat.f_oneway(sampleA, sampleB)

Ax = np.linspace(sampleAMean - 3*sampleAStd, sampleAMean + 3*sampleAStd, 100)

Bx = np.linspace(sampleBMean - 3*sampleBStd, sampleBMean + 3*sampleBStd, 100)

fig,ax = plt.subplots()

ax.plot(Ax, 105*stat.norm.pdf(Ax, sampleAMean, sampleAStd), color = 'firebrick')
ax.plot(Ax, 135*stat.norm.pdf(Ax, sampleAMean, sampleBStd), color = 'dodgerblue')
ax.hist(sampleA, alpha=0.5, label='Sample A', color = 'firebrick')
ax.hist(sampleB, alpha=0.5, label='Sample B', color = 'dodgerblue')
ax.tick_params(
axis='both',
labelleft=False,
labelbottom = False)
plt.show()

plt.savefig("Exercise1.pdf")


### Ex 2


import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.stats as stat
```

```
ant = pd.read_csv("Files/anthropogenic.dat").dropna()
nat = pd.read_csv("Files/natural.dat").dropna()

antDF = pd.DataFrame.to_numpy(ant)
natDF = pd.DataFrame.to_numpy(nat)

antMean = np.mean(ant)
natMean = np.mean(nat)

antVar = np.var(ant)
natVar = np.var(nat)

antStd = np.std(ant)
natStd = np.std(nat)

pVal = stat.ttest_ind(ant, nat).pvalue
fTest = stat.f_oneway(ant, nat)

antX = np.linspace(antMean - 3*antStd, antMean + 3*antStd, 100)
natX = np.linspace(natMean - 3*natStd, natMean + 3*natStd, 100)


fig,ax = plt.subplots()

ax.plot(antX, 2100000*stat.norm.pdf(antX, antMean, antStd), color = 'firebrick')
ax.plot(natX, 6350000*stat.norm.pdf(natX, natMean, natStd), color = 'dodgerblue')

ax.hist(ant, alpha=0.5, color = 'firebrick')
ax.hist(nat, alpha=0.5, color = 'dodgerblue')

#ax.tick_params(axis='both', labelleft=False, labelbottom = False)

#plt.savefig("Exercise2.pdf")

fig,ax = plt.subplots()

ax.plot(antX, stat.norm.pdf(antX, antMean, antStd), color = 'firebrick')
ax.plot(natX, stat.norm.pdf(natX, natMean, natStd), color = 'dodgerblue')

#ax.tick_params(axis='both', labelleft=False, labelbottom = False)

#plt.savefig("Exercise2Gauss.pdf")

### Ex 3

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stat
import pandas as pd



ex3Data = pd.read_csv("Files/DAex3.dat", delimiter=' ').dropna(axis = 1).to_numpy()
varX = ex3Data[0:, 0]
varY = ex3Data[0:, 1]
varZ = ex3Data[0:, 2]
```

```
xySlope, xyInt, xyCor, xyP, xyStd = stat.linregress(varX, varY)
xzSlope, xzInt, xzCor, xzP, xzStd = stat.linregress(varX, varZ)


Xarray = np.linspace(min(varX)-5, max(varX)+5, 1000)
XYarray = xySlope*Xarray + xyInt
XZarray = xzSlope*Xarray + xzInt

fig, ax = plt.subplots()
ax.plot(Xarray, XYarray, color = 'orange')
ax.plot(Xarray, XZarray, color = 'seagreen')
ax.scatter(varX, varY, color = 'firebrick', label = 'X vs Y')
ax.scatter(varX, varZ, color = 'dodgerblue', label = 'X vs Z')

plt.show()

plt.savefig('Exercise3.pdf')

### Ex 4

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stat


n = 100                # number of data points
x = np.random.ranf(n)*15-5;    # random x data values in range -5 to 10 uniformly distibuted
xx = np.arange(-5.5,10.501,0.01)   # x vector to use for plotting
p = [1, -10, -10, 20]     # polynomial coefficients


y = np.polyval(p, x)       # y data values
sigma = np.random.rand(n)*25;   # random uniformly distributed uncertainties in range 0 to 25
error=np.random.normal(0.0, 1.0, n)*sigma # add gaussian-distributed random noise
y=y+error

realY = np.polyval(p, xx)

coeff, cov = np.polyfit(x, y, 3, cov = True)
fitY = np.polyval(coeff, xx)

chi, pVal = stat.chisquare(fitY, realY)


plt.figure()

plt.errorbar(x, y, xerrr=None, yerr=sigma, linestyle='none', marker='*', color = 'orange')
plt.plot(xx, realY, color = 'firebrick', linewidth = 3)
plt.plot(xx, fitY, color = 'dodgerblue', linewidth = 3)

plt.savefig("Exercise4.pdf")

### Ex

import matplotlib.pyplot as plt
```

```python
import numpy as np
import scipy.stats as stat
import pandas as pd
import scipy.optimize
import pylab as plb



osloData = pd.read_csv("Files/DAex5_data1.dat", delimiter = ' ', skipinitialspace = True,
    skiprows = lambda x: x in range(0, 21)).dropna(axis = 1).to_numpy()
canbData = pd.read_csv("Files/DAex5_data2.dat", delimiter = ' ', skipinitialspace = True,
    skiprows = lambda x: x in range(0, 21)).dropna(axis = 1).to_numpy()

osloYear, osloX = osloData[0:, 0].tolist(), osloData[0:, 1:].tolist()
canbYear, canbX = canbData[0:, 0].tolist(), canbData[0:, 1:].tolist()

osloTemp = []
for i in range(len(osloX)):
    osloTemp += osloX[i]
osloTemp = osloTemp[:-2]

canbTemp = []
for i in range(len(canbX)):
    canbTemp += canbX[i]
canbTemp = canbTemp[:-2]

###############################
## Sloped
###############################


def fit_sin(tt, yy):
    #Fit sin to the input time sequence, and return fitting parameters "amp", "omega",
        "phase", "offset", "freq", "period" and "fitfunc"
    tt = np.array(tt)
    yy = np.array(yy)
    ff = np.fft.fftfreq(len(tt), (tt[1]-tt[0])) # assume uniform spacing
    Fyy = abs(np.fft.fft(yy))
    guess_freq = abs(ff[np.argmax(Fyy[1:])+1]) # excluding the zero frequency "peak", which is
        related to offset
    guess_amp = np.std(yy) * 2.**0.5
    guess_offset = np.mean(yy)
    guess = np.array([guess_amp, 2.*np.pi*guess_freq, 0., guess_offset, 0])

    def sinfunc(t, A, w, p, c, b): return A * np.sin(w*t + p) + b*t + c
    popt, pcov = scipy.optimize.curve_fit(sinfunc, tt, yy, p0=guess)
    A, w, p, c, b = popt
    f = w/(2.*np.pi)
    fitfunc = lambda t: A * np.sin(w*t + p) + b*t + c
    return {"amp": A, "omega": w, "phase": p, "offset": c, "freq": f, "gradient": b, "period":
        1./f, "fitfunc": fitfunc, "maxcov": np.max(pcov), "rawres": (guess,popt,pcov)}



##### Oslo
```

```
osloRes = fit_sin(range(len(osloTemp)), osloTemp)
osloXX = np.linspace(0, len(osloTemp), 5000)
osloMonths = [i for i in range(0, len(osloTemp))]

fig, ax = plt.subplots()

ax.scatter(osloMonths, osloTemp, color = 'firebrick', alpha = 0.5, marker = '.')
ax.plot(osloXX, osloRes["fitfunc"](osloXX), color = 'orange', linewidth = 1, alpha = 0.5)
plt.xlabel("Months Since 1817")
plt.ylabel("Average Temperature (deg C)")

plt.savefig("Figures/Exercise5OsloSlopedFull.svg")

fig, ax = plt.subplots()

ax.scatter(osloMonths, osloTemp, color = 'firebrick', alpha = 0.5, marker = '.')
ax.plot(osloXX, osloRes["fitfunc"](osloXX), color = 'orange', linewidth = 2, alpha = 0.7)
plt.xlabel("Months Since 1817")
plt.ylabel("Average Temperature (deg C)")
plt.xlim(450, 650)

plt.savefig("Figures/Exercise5OsloSlopedTrunc.svg")

# amp -10.679443548066253
# freq 0.08333615255723902
# gradient 0.0008706682279664362
# maxcov 0.007234390030615362
# offset 4.405784012586997
# omega 0.5236164893045206
# period 11.999594045491301
# phase 1.4894133785873147



##### Canberra

canbRes = fit_sin(range(len(canbTemp)), canbTemp)
canbXX = np.linspace(0, len(canbTemp), 5000)
canbMonths = [i for i in range(0, len(canbTemp))]

fig, ax = plt.subplots()

ax.scatter(canbMonths, canbTemp, color = 'dodgerblue', alpha = 0.5, marker = '.')
ax.plot(canbXX, canbRes["fitfunc"](canbXX), color = 'seagreen', linewidth = 2, alpha = 0.5)
plt.xlabel("Months since 1940")
plt.ylabel("Average Temperature (deg C)")

plt.savefig("Figures/Exercise5CanbSlopedFull.svg")

fig, ax = plt.subplots()

ax.scatter(canbMonths, canbTemp, color = 'dodgerblue', alpha = 0.5, marker = '.')
ax.plot(canbXX, canbRes["fitfunc"](canbXX), color = 'seagreen', linewidth = 2, alpha = 0.7)
plt.xlabel("Months since 1940")
plt.ylabel("Average Temperature (deg C)")
plt.xlim(450, 650)
```

```python
plt.savefig("Figures/Exercise5CanbSlopedTrunc.svg")

# amp 7.364528672121448
# freq 0.08333065708618924
# gradient 0.0012350109841513736
# maxcov 0.007057356744071391
# offset 12.586489795394588
# omega 0.5235819602415647
# period 12.0003853919656
# phase 1.5315102102125375




#############################
## Unsloped
#############################

def fit_sin(tt, yy):
    '''Fit sin to the input time sequence, and return fitting parameters "amp", "omega",
        "phase", "offset", "freq", "period" and "fitfunc"'''
    tt = np.array(tt)
    yy = np.array(yy)
    ff = np.fft.fftfreq(len(tt), (tt[1]-tt[0])) # assume uniform spacing
    Fyy = abs(np.fft.fft(yy))
    guess_freq = abs(ff[np.argmax(Fyy[1:])+1]) # excluding the zero frequency "peak", which is
        related to offset
    guess_amp = np.std(yy) * 2.**0.5
    guess_offset = np.mean(yy)
    guess = np.array([guess_amp, 2.*np.pi*guess_freq, 0., guess_offset])

    def sinfunc(t, A, w, p, c): return A * np.sin(w*t + p) + c
    popt, pcov = scipy.optimize.curve_fit(sinfunc, tt, yy, p0=guess)
    A, w, p, c = popt
    f = w/(2.*np.pi)
    fitfunc = lambda t: A * np.sin(w*t + p) + c
    return {"amp": A, "omega": w, "phase": p, "offset": c, "freq": f, "period": 1./f,
        "fitfunc": fitfunc, "maxcov": np.max(pcov), "rawres": (guess,popt,pcov)}


#####

osloRes = fit_sin(range(len(osloTemp)), osloTemp)
osloXX = np.linspace(0, len(osloTemp), 5000)


fig, ax = plt.subplots()

ax.scatter(osloMonths, osloTemp, color = 'firebrick', alpha = 0.5, marker = '.')
ax.plot(osloXX, osloRes["fitfunc"](osloXX), color = 'orange', linewidth = 2, alpha = 0.5)
plt.xlabel("Months Since 1817")
plt.ylabel("Average Temperature (deg C)")

plt.savefig("Figures/Exercise5OsloFull.svg")
```

```
fig, ax = plt.subplots()

ax.scatter(osloMonths, osloTemp, color = 'firebrick', alpha = 0.5, marker = '.')
ax.plot(osloXX, osloRes["fitfunc"](osloXX), color = 'orange', linewidth = 2, alpha = 0.7)
plt.xlabel("Months Since 1817")
plt.ylabel("Average Temperature (deg C)")
plt.xlim(450, 650)

plt.savefig("Figures/Exercise5OsloTrunc.svg")


# amp -10.681651355305279
#freq 0.0833361993728413
#maxcov 0.003910412379842344
#offset 5.412715328095888
#omega 0.5236167834556251
#period 11.999587304504471
#phase 1.4888928930197116







#####

canbRes = fit_sin(range(len(canbTemp)), canbTemp)
canbXX = np.linspace(0, len(canbTemp), 5000)

fig, ax = plt.subplots()

ax.scatter(canbMonths, canbTemp, color = 'dodgerblue', alpha = 0.5, marker = '.')
ax.plot(canbXX, canbRes["fitfunc"](canbXX), color = 'seagreen', linewidth = 2, alpha = 0.5)
plt.xlabel("Months since 1940")
plt.ylabel("Average Temperature (deg C)")

plt.savefig("Figures/Exercise5CanbFull.svg")

fig, ax = plt.subplots()

ax.scatter(canbMonths, canbTemp, color = 'dodgerblue', alpha = 0.5, marker = '.')
ax.plot(canbXX, canbRes["fitfunc"](canbXX), color = 'seagreen', linewidth = 2, alpha = 0.7)
plt.xlabel("Months since 1940")
plt.ylabel("Average Temperature (deg C)")
plt.xlim(450, 650)

plt.savefig("Figures/Exercise5CanbTrunc.svg")

# amp 7.361485838713342
# freq 0.08333038558950709
# maxcov 0.0037435118122164687
# offset 13.103356014538988
# omega 0.5235802543776005
# period 12.000424490126438
# phase 1.532599955534938
```

```
### Ex 6


import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stat
import pandas as pd
import scipy.optimize
import pylab as plb


samplingFreq = [i for i in range(1, 140)]
peakPos = []

for rate in samplingFreq:
    fs = rate # sampling frequency (Hz)

    ts = 1/fs # set sampling rate and interval
    period=10.0 #sampling period
    nfft = period/ts # length of DFT
    t=np.arange(0,period,ts)
    #h = ((1.7)*np.sin((2*np.pi*35.0*t)-0.6) + (2.5)*np.random.normal(0.0,1.0,t.shape))
    h = ((1.3)*np.sin(2*np.pi*5.0*t) + (1.7)*np.sin((2*np.pi*35.0*t)-0.6) +
        (2.5)*np.random.normal(0.0,1.0,t.shape))
    # combination of a 5 Hz signal a 35Hz signal and Gaussian noise

    H = np.fft.fft(h) # determine the Discrte Fourier Transform


    # Take the magnitude of fft of H
    mx = abs(H[0:np.int(nfft/2)])*(2.0/nfft) # note only need to examien first half of spectrum


    # Frequency vector
    f = np.arange(0,np.int(nfft/2))*fs/nfft

    mxList = list(mx)
    maxAmp = max(mxList)
    maxAmpInd = mxList.index(maxAmp)
    fAtMaxAmp = f[maxAmpInd]

    peakPos.append(fAtMaxAmp)



plt.figure(1)
plt.scatter(samplingFreq, peakPos, color = 'firebrick', marker = 'x')
#plt.ylabel("Sampling Frequency (Hz)")
#plt.xlabel("Peak Position")
plt.savefig('Figures/Exercise6PeakGuess.pdf')


#plt.figure(2)
#plt.plot(t,h);
#plt.title('Sine Wave Signals');
```

```
#plt.xlabel('Time (s)');
#plt.ylabel('Amplitude');

samplingFreq = [75]
for rate in samplingFreq:
    fs = rate # sampling frequency (Hz)

    ts = 1/fs # set sampling rate and interval
    period=10.0 #sampling period
    nfft = period/ts # length of DFT
    t=np.arange(0,period,ts)
    h = ((1.7)*np.sin((2*np.pi*35.0*t)-0.6) + (2.5)*np.random.normal(0.0,1.0,t.shape))
    #h = ((1.3)*np.sin(2*np.pi*5.0*t) + (1.7)*np.sin((2*np.pi*35.0*t)-0.6) +
        (2.5)*np.random.normal(0.0,1.0,t.shape))
    # combination of a 5 Hz signal a 35Hz signal and Gaussian noise

    H = np.fft.fft(h) # determine the Discrte Fourier Transform


    # Take the magnitude of fft of H
    mx = abs(H[0:np.int(nfft/2)])*(2.0/nfft) # note only need to examien first half of spectrum


    # Frequency vector
    f = np.arange(0,np.int(nfft/2))*fs/nfft


    plt.figure(3)
    plt.plot(f,mx, color = 'firebrick')
    #plt.title('Amplitude Spectrum of Sine Wave signals');
    #plt.xlabel('Frequency (Hz)');
    #plt.ylabel('Amplitude');
    plt.xlim(0, max(samplingFreq)/2)
    #plt.ylim(1, 2.5)
    plt.savefig('Figures/Exercise6Amp35.pdf')

### Ex 7

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stat
import pandas as pd
import scipy.optimize
import pylab as plb

data = pd.read_csv("Files/DAex7data.txt", delimiter = ' ').dropna(axis = 1).to_numpy()
days, horizVel = data[:, 0], data[:, 1]

plt.subplot(2,1,1)
plt.plot(days, horizVel, color = 'black')


# Fourier transformation converting horizVel to the frequency domain.
hvFT = np.fft.fft(horizVel)
dFT = days
```

```
# Scaling horizVel vector by (len(horizVel)).
hvFT = abs(hvFT) * 2/256

# Scaling x by (len(horizVel))/fs
# fs = sampling frequency = 240 hours = 10 days*24h each.
dFT *= 256/240



plt.subplot(2,1,2)

plt.axvline(12, color='firebrick', label='Semi-Diurnal Tide') # 12 Hours
plt.axvline(23.5, color='dodgerblue', label='Diurnal Tide') # 24 Hours
# Logarithmic x-axis
plt.xscale('log')
plt.plot(dFT, hvFT, color='black')

plt.savefig('Figures/Exercise7.pdf')
```