Laboratory 4: Fourier Analysis using Python

ismisebrendan

April 18, 2023

1 Introduction

1.1 Fourier Series

A Fourier series can be used to find the underlying signals in a periodic signal as a sum of sin and cosine functions. All periodic functions with period $T = 2\pi/\omega$ can be represented by a Fourier series

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + b_n \sin(n\omega t) \tag{1.1}$$

The series can theoretically contain an infinite series (although in practice it is only evaluated up to some finite N) of harmonic functions with frequencies $\omega_n = n\omega_1$ where n is a positive integer. The coefficients a_n and b_n represent the amount of the harmonic functions in the series.

The values of a_0, a_n and b_n are given below,

$$a_0 = \frac{1}{T}\int_0^T f(t)dt \tag{1.2}$$

$$a_n = \frac{2}{T}\int_0^T f(t)\cos(n\omega t)dt \tag{1.3}$$

$$b_n = \frac{2}{T}\int_0^T f(t)\sin(n\omega t)dt \tag{1.4}$$

of course a_n and b_n must be evaluated individually for each value of n. And when evaluated they give discrete points in a graph against n. This makes computerised methods for calculating Fourier series very useful as they can easily repeat themselves multiple times.

1.2 Square and Rectangular Waves

A pulse wave is a periodic function in which the amplitude of the function alternates between a maximum and minimum value with the transitions between maxima and minima being (ideally) instantaneous. Both square and rectangular waves are examples of pulse waves. In a square wave half of the time is spent at the maximum while half is spent at the minimum. Any wave which does not spend exactly half its time at both the maximum and minimum is referred to as a rectangular wave.

Square and rectangular waves can be formed from the superposition of many sin and cos functions as in a Fourier series.

As the square wave is an odd function all a_n are equal to 0. This is a general property of Fourier series of odd functions

$$a_n = 0 \quad n = 1, 2, 3, ... \tag{1.5}$$

The b_n coefficients can be found to equal,

$$b_n = \begin{cases} \frac{4}{\pi n} & n = 1, 3, 5, ... \\ 0 & n = 2, 4, 6, ... \end{cases} \tag{1.6}$$

This is shown in *Appendix A: Derivations, 5.1.1 Derivation of Fourier Coefficients for the Square Wave*

The coefficients of the Fourier series of the rectangular wave must be calculated as shown in *Appendix A: Derivations, 5.1.2 Derivation of Fourier Coefficients for the Rectangular Wave* to give

$$a_0 = \frac{1 - \pi}{\pi} \approx -0.6817 \tag{1.7}$$

$$a_n = \frac{2\sin(n)}{\pi n} \tag{1.8}$$

$$b_n = \frac{2 - 2\cos(n)}{\pi n} \tag{1.9}$$

1.3 Fourier Transforms

Fourier transforms are used to express aperiodic functions in terms of cosine and sine functions as an integral over a continuous range of frequencies. Fourier transforms produce complex functions given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{1.10}$$

This can be undone by a so-called 'back transform' to reproduce the original function,

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \tag{1.11}$$

In practice what is often done is samples are taken of a signal are taken at regular intervals and a discrete Fourier transform is performed on the data. Say a time dependent signal represented by a function $f(t)$ is sampled N times at intervals h over the time interval $[0, h(N-1)]$. Let $t_m = mh$ where $m = 0, 1, 2, ..., N-1$.

It is assumed that the function is periodic with a period $\tau = Nh$, i.e. $f(t+\tau) = f(t)$ or $f(t_{m+N}) = f_m$. Under this assumption the fundamental frequency is $\omega_1 = 2\pi/\tau = 2\pi/Nh$. And the further frequencies are, as above, given by $\omega_n = n\omega_1$. This produces a discrete Fourier transform of the form,

$$F_n = \sum_{m=0}^{N-1} f_m e^{\frac{-2\pi imn}{N}} \tag{1.12}$$

Where $f_m = f(t_m)$. This can be expended into separate expressions for the real and imaginary terms of F_n to give the expression below in Eq. 1.13 for the real terms, and 1.14 for the imaginary terms.

$$F_{n_{real}} = \sum_{m=0}^{N-1} f_m \cos\left(\frac{2\pi mn}{N}\right) \tag{1.13}$$

$$F_{n_{imag}} = \sum_{m=0}^{N-1} -f_m \sin\left(\frac{2\pi mn}{N}\right) \tag{1.14}$$

The discrete Fourier 'back transform' is given by,

$$f_m = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{\frac{2\pi imn}{N}} \tag{1.15}$$

which can be expanded to give,

$$f_m = \frac{1}{N} \sum_{n=0}^{N-1} F_n \left(\cos\left(\frac{2\pi mn}{N}\right) + i \sin\left(\frac{2\pi mn}{N}\right)\right) \tag{1.16}$$

This allows expressions for the real and imaginary components of the back transform, $f_{m_{real}}$ and $f_{m_{imag}}$ respectively, to be calculated.

$$f_{m_{real}} = \frac{1}{N} \sum_{n=0}^{N-1} \left(Re\left[F_n\right] \cos\left(\frac{2\pi mn}{N}\right) - Im\left[F_n\right] \sin\left(\frac{2\pi mn}{N}\right)\right) \tag{1.17}$$

$$f_{m_{imag}} = \frac{1}{N}\sum_{n=0}^{N-1}\left(Re\left[F_n\right] i\sin\left(\frac{2\pi mn}{N}\right) + Im\left[F_n\right] i\cos\left(\frac{2\pi mn}{N}\right)\right) \tag{1.18}$$

It can be shown (see *Appendix A: Derivations, 5.1.3 Nyquist Symmetry*) that $F_{N-n} = F_n^*$ where F_n^* is the complex conjugate of F_n. Thus the highest frequency component which can be analysed is $F_{n/2-1}$ whose frequency is referred to as the Nyquist frequency and is given by,

$$\nu_n = \frac{\frac{N}{2} - 1}{Nh} \tag{1.19}$$

If the signal being analysed has a component with frequency greater than the Nyquist frequency, there are fewer than two sample points per period. This means that there will be one or more frequencies below the Nyquist frequency for which the amplitude equals the true amplitude at the sample points, and these incorrect lower frequencies will appear in the calculated spectrum.

The power spectrum of the discrete Fourier transform is a plot of the values

$$P_n = F_{n_{real}}^2 + F_{n_{imag}}^2 \tag{1.20}$$

1.4 Simpson's Rule

Simpson's rule is a method of numerical integration which uses parabolas to approximate the area under a curve and is more accurate than the trapezoidal rule.

The integral of a parabola of the form $f(x) = ax^2 + bx + c$ has an integral over the region $[-h, h]$ of the form

$$\int_{-h}^{+h} ax^2 + bx + cdx = \frac{h}{3}(2ah^2 + 6c) \tag{1.21}$$

The values of the function f at $x = -h$ (y_0), $x = 0$ (y_1) and $x = h$ (y_2) are

$$y_0 = ah^2 - bh + c \quad y_1 = c \quad y_2 = ah^2 + bh + c \tag{1.22}$$

and from these it is found that $2ah^2 = y_0 - 2y_1 + y_2$ and $c = y_1$ making the area under the curve on the interval $[-h, h]$

$$A_0 = \frac{h}{3}(2ah^2 + 6c) = \frac{h}{3}(y_0 + 4y_1 + y_2) \tag{1.23}$$

This can be expanded to the give area A_1 under an adjacent interval $[h, 3h]$ as

$$A_1 = \frac{h}{3}(y_2 + 4y_3 + y_4) \tag{1.24}$$

And the total area is

$$A_0 + A_1 = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + y_4) \tag{1.25}$$

If this is extended to the integral $\int_a^b f(x)dx$ with an even number n equal steps of length $h = \frac{b-a}{n}$ the integral can be approximated as

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right) \tag{1.26}$$

Where $x_j = a + jh$ for $j = 0, 1, ..., n$ and $x_0 = a$ and $x_n = b$. Larger values of n increase the accuracy of the integration

2 Methodology

2.1 Using Simpson's Rule to Evaluate Fourier Series

The code for this section can be found in *Appendix B: Code, 5.2.1 Using Simpson's Rule to Evaluate Fourier Series* below.

A function *Simpson* was defined which would integrate an inputted function using Simpson's Rule over an integral $[a, b]$, using an even number of steps $2n$. It takes an input of n which is then doubled in order to ensure only an even number of steps is used. It was tested using the integral $\int_0^1 e^x dx$ using $n = 8$ steps.

Functions *a0*, *ak* and *bk* were defined to find the Fourier coefficients as in Eqs. 1.2, 1.3 and 1.4 respectively.

The function *a0* simply defines the period, T, from the inputted fundamental frequency, ω, and calls the *Simpson* function to calculate the integral over the period using two times the inputted number of steps (again to ensure only an even number of steps are used). This value is then returned at the end of the function.

The functions *ak* and *bk* are slightly more complicated than *a0*. The function *ak* takes an extra positional argument k which is the number of Fourier coefficients to be calculated. It again defines the period T, from the inputted fundamental frequency, ω. Due to the method by which the *Simpson* function operates an extra "dummy function" which is to be integrated over is defined as $f(t)\cos(t\omega i)$ where i is the variable which is looped over to give values of a_k for different ks. The dummy function is integrated over one period using *Simpson* for all values of k from 1 to the inputted value. These different values of a_k are appended to an array which is returned at the end of the function.

The function *bk* is the same as *ak* with the exception that the dummy function is redefined to be $f(t)\sin(t\omega i)$.

These three functions were then combined in another function *fourierSeries* which calculates all of the Fourier coefficients for an inputted function as well as evaluating the Fourier series at an inputted value of t.

In order to reduce the length of the script two functions *graphs* and *coefficients* were defined. The function *graphs* was defined to produce a graph of a given function and

Fourier series together as well as a second plot of the Fourier coefficients as discrete points. Whereas *coefficients* was defined to output the Fourier coefficients as text.

The Fourier series of the following functions were then found;

- $f(t) = \sin(\omega t)$
- $f(t) = \cos(\omega t) + 3\cos(2\omega t) - 4\cos(3\omega t)$
- $f(t) = \sin(\omega t) + 3\sin(3\omega t) + 5\sin(5\omega t)$
- $f(t) = \sin(\omega t) + 2\cos(3\omega t) + 3\sin(5\omega t)$

for values of k ≤ 10, and $\omega = 1$. Their graphs as well as the graph of their Fourier series and coefficients were plotted using *graphs* and their Fourier coefficients were outputted using *coefficients*.

2.2 Finding the Fourier Series of Square and Rectangular Waves

The code for this section can be found in *Appendix B: Code, 5.2.2 Finding the Fourier Series of Square and Rectangular Waves* below.

The same functions *Simpson*, *a0*, *ak*, *bk* and *fourierSeries* were defined in a new file to allow the calculation of a Fourier series.

A new function *square* was then defined to plot a square wave over any range using a step function as shown below,

$$f(\theta) = \begin{cases} 1 & 0 \leq \theta \leq \pi \\ -1 & \pi < \theta \leq 2\pi \end{cases} \tag{2.1}$$

where $\theta = \omega t$.

It initially takes the inputted value and multiplies it by ω to give θ as defined above, and then initialises an empty array, *step_list* to be used for the output. Another function was then defined within this, *squareFunc*. This function is the main one concerned with the assignment of the inputted value to the correct output in the step function.

The inputted value to *squareFunc* is first verified to be in the interval $[0, 2\pi]$. If it is not then $2\pi\frac{|\theta|}{\theta}$ is subtracted from it until $|\theta| < 2\pi$.

As this is an odd function, $f(-\theta) = -f(\theta)$, and Eq. 2.1 can be rewritten as,

$$f(\theta) = \begin{cases} 1 & -2\pi \leq \theta < \pi, \quad 0 \leq \theta \leq \pi \\ -1 & -\pi \leq \theta < 0, \quad \pi < \theta \leq 2\pi \end{cases} \tag{2.2}$$

then, based on what θ has been adjusted to the correct value is returned.

Back in *square* the input is checked to determine if it is an array or number (float or int). If it is a number then the output of *squareFunc* is appended to*step_list*. If instead it is an array, the elements are looped over and this is done for each element in the array. The data types must be separated because floats and ints can't be iterated over and because when checking the value of the input in *squareFunc* it does not separate

the values and attempts to determine the truth value of the entire array, which is not possible.

Finally the array *step_list* is returned.

The Fourier series of the function was then calculated using *fourierSeries* as above. This was plotted alongside the square wave from *sqaure* for different numbers of coefficients; 1, 2, 3, 5, 10, 20 and 30 over the interval $[0, 2\pi]$. This was done both individually and with all of the plots overlaid on top of each other. A plot of the function and Fourier Series including 30 coefficients over the interval $[-10, 10]$ was also made.

A function *rectangular* was then defined in a very similar way to *square* with the graph depicting the function,

$$f(\theta) = \begin{cases} 1 & 0 \leq \omega t \leq \omega\tau \\ -1 & \omega\tau < \omega t \leq 2\pi \end{cases} \tag{2.3}$$

or

$$f(\omega t) = \begin{cases} 1 & -2\pi \leq \omega t < -2\pi + \omega\tau, \quad 0 \leq \omega t \leq \omega\tau \\ -1 & -2\pi + \omega\tau \leq \omega t < 0, \quad \omega\tau < \omega t \leq 2\pi \end{cases} \tag{2.4}$$

The same graphs were generated for this rectangular function with $\tau = 1$ as were for the square function.

2.3 The Discrete Fourier Transform in Python

The code for this section can be found in *Appendix B: Code, 5.2.3 The Discrete Fourier Transform in Python* below.

A new python file was created and the functions *fReal*, *fImag*, *fmReal* and *fmImag* which correspond to Eqs. 1.13, 1.14, 1.17 and 1.18 respectively.

The function $f(t) = sin(0.45\pi t)$ was analysed first, using the functions *fReal* and *fImag*. The number of samples to be taken, N, was set to 128. The term f_m in Eqs. 1.13, 1.14 and 1.16 are shorthand for $f(t_m)$, the points at which the signal is sampled, where $t_m = mh$ and $m = 0, 1, 2, ..., N - 1$ and h is the interval over which samples are taken.

The function and the sampled points were plotted for a time $\tau = Nh$. The Fourier components $F_{n,real}$ and $F_{n,imag}$ were then plotted as functions of n.

The real and imaginary components of the back transform of the function and the original function were plotted on the same graph and the difference between the original function and the real component of the back transform was also plotted plotted.

This was done for the function $f(t) = sin(0.45\pi t)$ with sampling intervals $h = 0.1$ and $h = 5/144$.

The above was then repeated for the function $f(t) = cos(6\pi t)$, with N equal to 32 and h set to 0.6, 0.2, 0.1 and 0.04. The same graphs were generated, along with an extra graph of the power spectrum for each value of h.

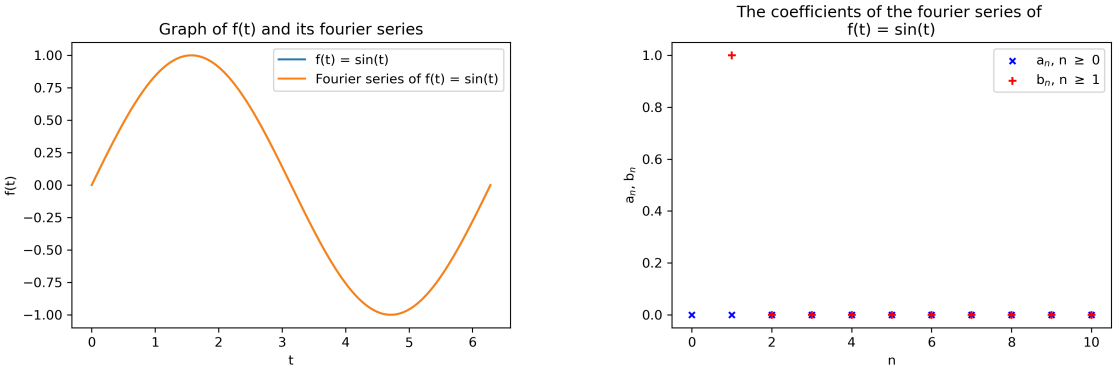
3 Results

3.1 Using Simpson's Rule to Evaluate Fourier Series

Simpson's rule found that $\int_0^1 e^x dx = 1.718284154699897$ which deviates from the actual solution of $e - 1 \approx 1.718281828459045$ by only $2.326240851 \times 10^{-6}$ showing that the function *Simpson* is good at approximating the integration of functions.

3.1.1 $f(t) = \sin(t)$

The graph of the function with its Fourier series overlaid on it as well as the plot of its Fourier coefficients is shown below in figure 3.1.



(a) The graph of $f(t) = \sin(t)$ with its Fourier series

(b) The Fourier coefficients of $f(t) = \sin(t)$ up to $n = 10$

Figure 3.1: The graph and Fourier coefficients of the function $f(t) = \sin(t)$

n	a_n	b_n
0	-3.356×10^{-17}	
1	3.650×10^{-17}	1.000
2	2.170×10^{-17}	-5.428×10^{-17}
3	-2.518×10^{-17}	-1.295×10^{-16}
4	-1.532×10^{-16}	8.635×10^{-17}
5	2.265×10^{-16}	-9.375×10^{-17}
6	-9.144×10^{-18}	1.308×10^{-16}
7	1.460×10^{-17}	1.431×10^{-16}
8	1.472×10^{-16}	-7.772×10^{-17}
9	7.458×10^{-16}	-6.476×10^{-16}
10	-1.038×10^{-17}	-5.230×10^{-16}

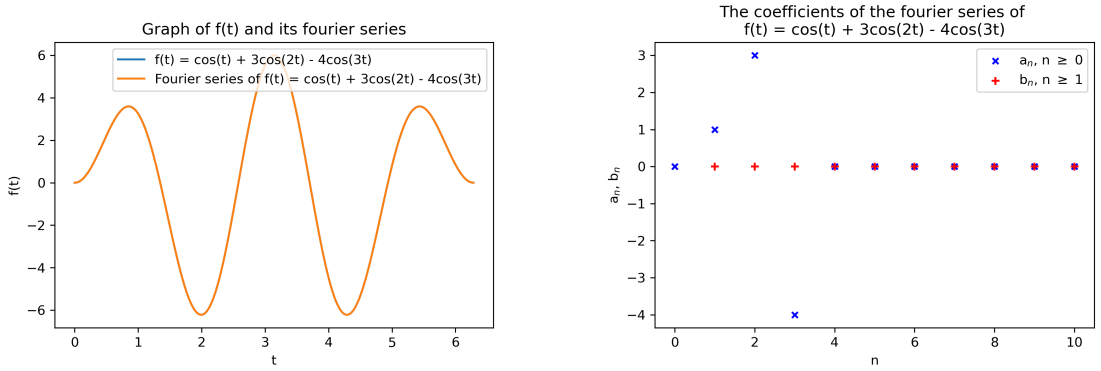
Table 3.1: The Fourier coefficients of the function $f(t) = \sin(t)$

The Fourier coefficients (truncated to 4 significant figures) are tabulated in table 3.1. As can be seen obviously all of the values except that of b_1 are extremely small, and in

fact if the coefficients were to be found analytically they would be found to all equal 0, except for b_1 which would indeed be found to equal 1. This is because the Fourier series of a function is expressed as in Eq. 1.1 above. In this case $\omega = 1$ and so the only term which should be non-zero is the coefficient of the $sin(t)$ term, i.e. the sin term for when $n = 1$. Which is b_1, and should be equal to 1.

3.1.2 $f(t) = \cos(t) + 3\cos(2t) - 4\cos(3t)$

The graph of the function with its Fourier series overlaid on it as well as the plot of its Fourier coefficients is shown below in figure 3.2.



(a) The graph of $f(t) = \cos(t) + 3\cos(2t) - 4\cos(3t)$ with its Fourier series

(b) The Fourier coefficients of $f(t) = \cos(t) + 3\cos(2t) - 4\cos(3t)$ up to $n = 10$

Figure 3.2: The graph and Fourier coefficients of the function $f(t) = \cos(t) + 3\cos(2t) - 4\cos(3t)$

n	a_n	b_n
0	-5.033×10^{-16}	
1	1.000	2.245×10^{-16}
2	3.000	2.911×10^{-16}
3	-4.000	2.763×10^{-16}
4	6.587×10^{-16}	-6.415×10^{-17}
5	9.943×10^{-16}	3.010×10^{-16}
6	-9.227×10^{-16}	-1.485×10^{-15}
7	-6.711×10^{-16}	-2.958×10^{-15}
8	-1.310×10^{-15}	3.940×10^{-15}
9	7.327×10^{-16}	-4.194×10^{-16}
10	1.086×10^{-16}	-1.239×10^{-15}

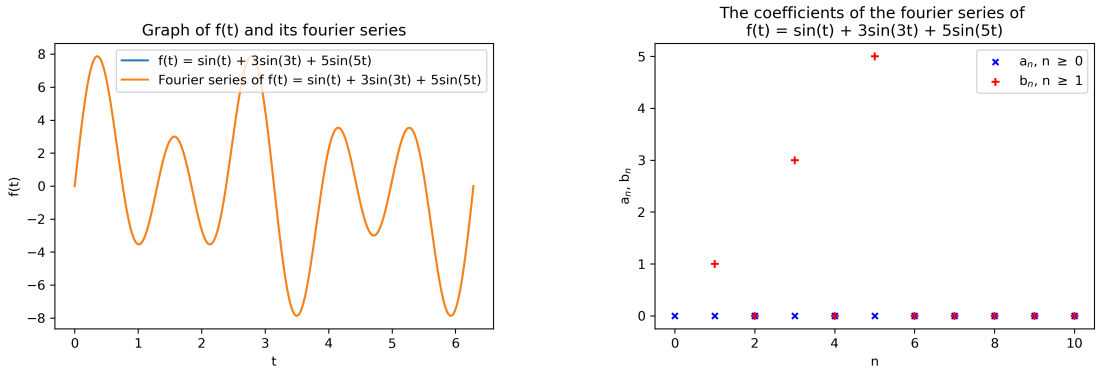
Table 3.2: The Fourier coefficients of the function $f(t) = \cos(t) + 3\cos(2t) - 4\cos(3t)$

The Fourier coefficients (truncated to 4 significant figures) are tabulated in table 3.2. As can be seen obviously all of the values except those of a_1, a_2 and a_3 are extremely

small, and in fact if the other coefficients were to be found analytically they would be found to all equal 0. From the same logic as that applied above to the coefficients of the $f(t) = sin(t)$ it can be reasoned that the coefficients of the cos(t) (n=1), cos(2t) (n=2) and cos(3t) (n=3) i.e. a_1, a_2 and a_3 should be 1, 3 and -4 respectively.

3.1.3 $f(t) = \sin(t) + 3\sin(3t) + 5\sin(5t)$

The graph of the function with its Fourier series overlaid on it as well as the plot of its Fourier coefficients is shown below in figure 3.3.



(a) The graph of $f(t) = \sin(t) + 3\sin(3t) + 5\sin(5t)$ and its Fourier series

(b) The Fourier coefficients of $f(t) = \sin(t) + 3\sin(3t) + 5\sin(5t)$ up to $n = 10$

Figure 3.3: The graph and Fourier coefficients of the function $f(t) = \sin(t) + 3\sin(3t) + 5\sin(5t)$

n	a_n	b_n
0	6.179×10^{-17}	
1	1.828×10^{-16}	1.000
2	-2.613×10^{-16}	-1.332×10^{-16}
3	-5.870×10^{-16}	3.000
4	-2.761×10^{-16}	-5.329×10^{-16}
5	6.004×10^{-16}	5.000
6	-5.722×10^{-16}	8.290×10^{-16}
7	-3.797×10^{-16}	-1.332×10^{-15}
8	-4.685×10^{-16}	2.368×10^{-16}
9	-5.129×10^{-16}	-7.401×10^{-17}
10	9.568×10^{-16}	-2.819×10^{-15}

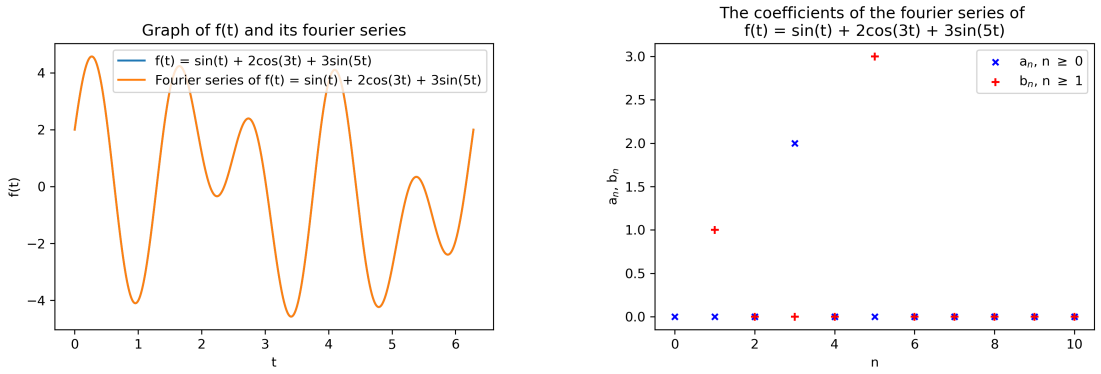
Table 3.3: The Fourier coefficients of the function $f(t) = \sin(t) + 3\sin(3t) + 5\sin(5t)$

The Fourier coefficients (truncated to 4 significant figures) are tabulated in table 3.3. As can be seen obviously all of the values except those of b_1, b_3 and b_5 are extremely small, and in fact if the other coefficients were to be found analytically they would be

found to all equal 0. From the same logic as that applied above to the coefficients of the $f(t) = sin(t)$ it can be reasoned that the coefficients of the $\sin(t)$ (n=1), $\sin(3t)$ (n=3) and $\sin(5t)$ (n=5) i.e. b_1, b_3 and b_5 should be 1, 3 and 5 respectively.

3.1.4 $f(t) = \sin(t) + 2\cos(3t) + 3\sin(5t)$

The graph of the function with its Fourier series overlaid on it as well as the plot of its Fourier coefficients is shown below in figure 3.4.



(a) The graph of $f(t) = \sin(t) + 2\cos(3t) + 3\sin(5t)$ and its Fourier series

(b) The Fourier coefficients of $f(t) = \sin(t) + 2\cos(3t) + 3\sin(5t)$ up to $n = 10$

Figure 3.4: The graph and Fourier coefficients of the function $f(t) = \sin(t) + 2\cos(3t) + 3\sin(5t)$

n	a_n	b_n
0	-1.702×10^{-16}	
1	-1.850×10^{-16}	1.000
2	-2.146×10^{-16}	-5.226×10^{-18}
3	2.000	-6.150×10^{-17}
4	-5.033×10^{-16}	-1.400×10^{-16}
5	3.516×10^{-16}	3.000
6	-1.673×10^{-15}	1.228×10^{-15}
7	1.380×10^{-15}	6.164×10^{-16}
8	-3.627×10^{-16}	-8.012×10^{-17}
9	-4.811×10^{-17}	7.888×10^{-16}
10	1.776×10^{-16}	-1.920×10^{-15}

Table 3.4: The Fourier coefficients of the function $f(t) = \sin(t) + 2\cos(3t) + 3\sin(5t)$

The Fourier coefficients (truncated to 4 significant figures) are tabulated in table 3.4. As can be seen obviously all of the values except those of b_1, a_3 and b_5 are extremely small, and in fact if the other coefficients were to be found analytically they would be found to all equal 0. From the same logic as that applied above to the coefficients of the

$f(t) = sin(t)$ it can be reasoned that the coefficients of the sin(t) (n=1), cos($3t$) (n=3) and sin($5t$) (n=5) i.e. b_1, a_3 and b_5 should be 1, 3 and 5 respectively.

3.2 Finding the Fourier Series of Square and Rectangular Waves

3.2.1 The Square Wave

The square wave was plotted over the interval $[0, 2\pi]$ and its Fourier series was evaluated. The graph of the function with its Fourier series for various values of n is shown in figure 3.5

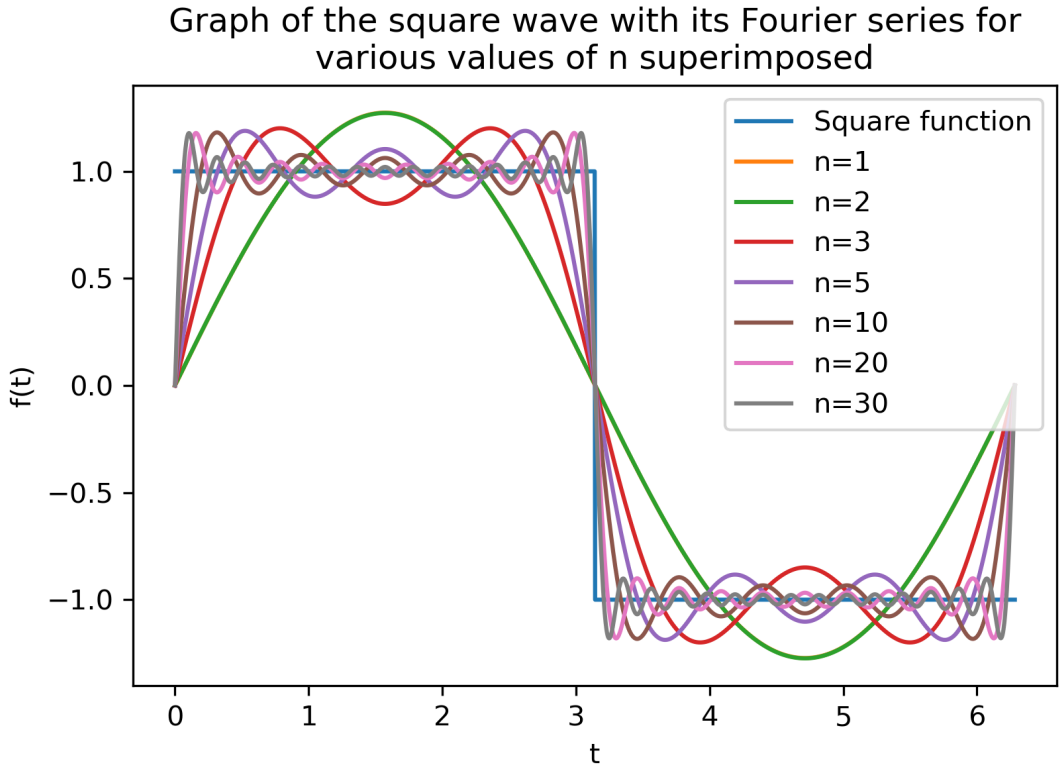


Figure 3.5: The graph of the square wave function with its Fourier series evaluated for $n = 1, 2, 3, 5, 10, 20, 30$.

Figure 3.5 shows that as n increases the Fourier series becomes a better approximation of the function. This is also clearly seen in figures 5.1a to 5.1g (available in *5.3 Appendix C: Extra Graphs* below) which show the Fourier series for various values of n with each plotted separately.

As expected, as n increases the Fourier series better approximates the square wave. Noticeably the graphs for $n = 1$ and $n = 2$ are the same, this is because as stated in

Eq. 1.6 above $b_2 = 0$ so there is no difference between the sine terms Fourier series for $n = 1$ and $n = 2$ and as this is an odd function all $a_n = 0$.

Table 3.5 below shows the numerical values of the coefficients found and analytic values which were expected according to Eqs. 1.5 and 1.6. As was expected the b_n for even n were very small and almost equal to 0, while the b_n for odd n were all equal to the expected values up to four significant figures except for b_3 and b_{29} (which differ from their expected values by only 0.09% and 0.02% respectively). The values of a_n which were expected to all be equal to 0 were found to be very small, with $a_0 = 0.0006667$ and all other a_n being either -2 or 2 times that value. These are all very small, yes, however they should ideally be even smaller which may be possible for larger number of steps of integration in Simpson's rule.

Table 3.5: The values calculated for the Fourier coefficients for the square wave up to $n = 30$

n	Numerical results		Analytical results	
	a_n	b_n	a_n	b_n
0	0.0006667		0	
1	−0.001333	1.273	0	1.273
2	0.001333	-5.995×10^{-17}	0	0
3	−0.001333	0.4248	0	0.4244
4	0.001333	-2.670×10^{-17}	0	0
5	−0.001333	0.2546	0	0.2546
6	0.001333	2.531×10^{-17}	0	0
7	−0.001333	0.1819	0	0.1819
8	0.001333	-5.037×10^{-17}	0	0
9	−0.001333	0.1415	0	0.1415
10	0.001333	-9.344×10^{-17}	0	0
11	−0.001333	0.1157	0	0.1157
12	0.001333	-7.798×10^{-17}	0	0
13	−0.001333	0.09794	0	0.09794
14	0.001333	4.847×10^{-17}	0	0
15	−0.001333	0.08488	0	0.08488
16	0.001333	3.022×10^{-17}	0	0
17	−0.001333	0.07490	0	0.07490
18	0.001333	1.241×10^{-16}	0	0
19	−0.001333	0.06701	0	0.06701
20	0.001333	-6.201×10^{-19}	0	0
21	−0.001333	0.06063	0	0.06063
22	0.001333	1.008×10^{-16}	0	0
23	−0.001333	0.05536	0	0.05536
24	0.001333	1.388×10^{-16}	0	0
25	−0.001333	0.05093	0	0.05093

26	0.001333	-7.354×10^{-17}	0	0
27	−0.001333	0.04716	0	0.04716
28	0.001333	3.632×10^{-17}	0	0
29	−0.001333	0.04391	0	0.04390
30	0.001333	3.436×10^{-17}	0	0

As a matter of curiosity the square wave and its Fourier series for $n = 30$ on the interval $[-10, 10]$ is plotted in figure 5.2 in *Appendix C* below.

3.2.2 The Rectangular Wave

The rectangular wave was plotted over the interval $[0, 2\pi]$ and its Fourier series was evaluated. The graph of the function with its Fourier series for various values of n is shown in figure 3.6

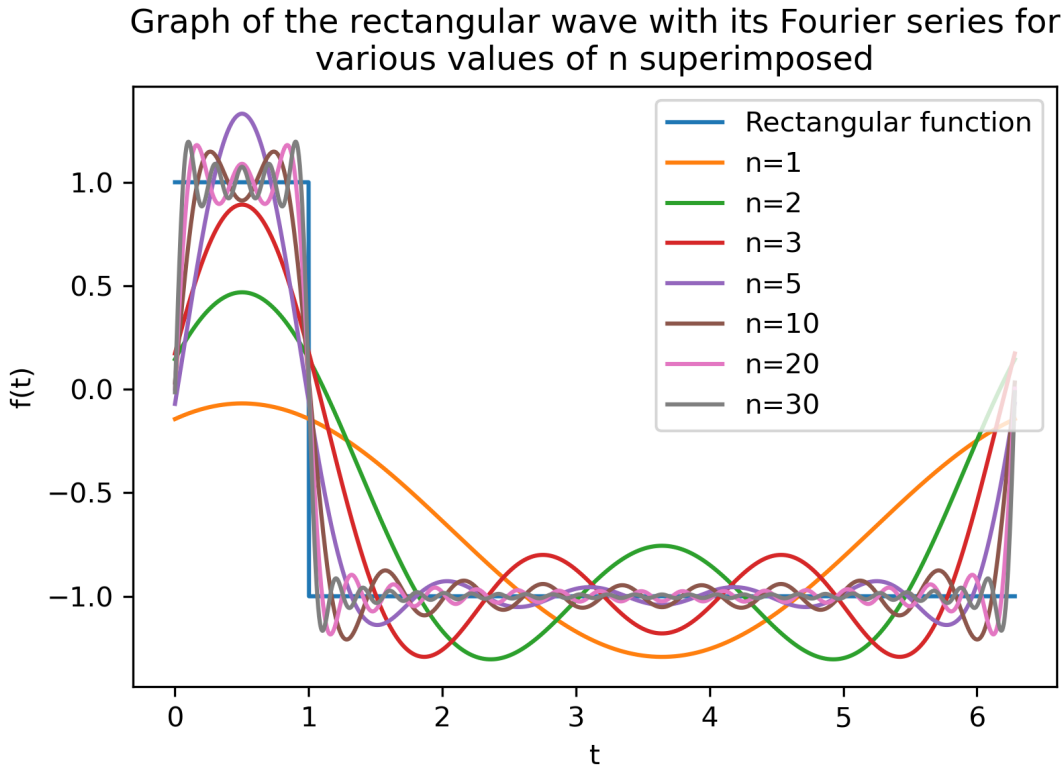


Figure 3.6: The graph of the rectangular wave function with its Fourier series evaluated for $n = 1, 2, 3, 5, 10, 20, 30$.

Figure 3.6 shows that as n increases the Fourier series becomes a better approximation of the function. This is also clearly seen in figures 5.4a to 5.4g (available in *5.3 Appendix C: Extra Graphs* below) which show the Fourier series for various values of n with each

plotted separately. As expected, as n increases the Fourier series better approximates the square wave.

Table 3.7 below shows the numerical values of the coefficients found and analytic values which were expected according to Eqs. 1.7, 1.8 and 1.9. The numerical values of all a_n and b_n were very similar their analytical values except for b_{25} which was found to be 2.602×10^{-18} significantly smaller than the expected analytical value of 2.240×10^{-4}. This number was calculated using 1000 steps in Simpson's Rule, however if the number of steps is increased to 10,000 b_{25} is found to equal 2.175×10^{-4} which is much closer to the analytical solution showing that it is simply a numerical error which occurs for this value of n.

Table 3.7: The values calculated for the Fourier coefficients for the rectangular wave up to $n = 30$ using 1000 steps in the integration according to Simpson's rule.

n	Numerical results		Analytical results	
	a_n	b_n	a_n	b_n
0	−0.6807		−0.6817	
1	0.5368	0.2944	0.5357	0.2927
2	0.2886	0.4526	0.2894	0.4508
3	0.02792	0.4226	0.02995	0.4223
4	−0.1218	0.2616	−0.1204	0.2632
5	−0.1215	0.08925	−0.1221	0.09121
6	−0.02768	0.003665	−0.02964	0.004226
7	0.06128	0.02374	0.05975	0.02238
8	0.07842	0.09318	0.07873	0.09116
9	0.02728	0.1360	0.02915	0.1352
10	−0.03634	0.1159	−0.03463	0.1171
11	−0.05784	0.05557	−0.05787	0.05762
12	−0.02673	0.007204	−0.02847	0.008284
13	0.02242	0.005415	0.02058	0.004532
14	0.04530	0.04129	0.04505	0.03926
15	0.02603	0.07599	0.02760	0.07468
16	−0.01341	0.07727	−0.01146	0.07789
17	−0.03654	0.04578	−0.03600	0.04775
18	−0.02518	0.01050	−0.02656	0.01201
19	0.007041	0.0007211	0.005022	0.0003785
20	0.02986	0.02073	0.02906	0.01884
21	0.02421	0.04861	0.02536	0.04692
22	−0.002304	0.05781	−0.0002561	0.05787
23	−0.02447	0.04067	−0.02342	0.04243
24	−0.02311	0.01344	−0.02402	0.01527
25	−0.001333	2.602×10^{-18}	−0.003370	0.0002240
26	0.01996	0.01024	0.01867	0.008645

27	0.02190	0.03241	0.02255	0.03047
28	0.004172	0.04513	0.006159	0.04462
29	−0.01606	0.03697	−0.01457	0.03837
30	−0.02059	0.01593	−0.02097	0.01795

As a matter of curiosity the rectangular wave and its Fourier series for $n = 30$ on the interval $[-10, 10]$ is plotted in figure 5.3 in *Appendix C* below.

3.3 The Discrete Fourier Transform in Python

3.3.1 $f(t) = sin(0.45\pi t)$

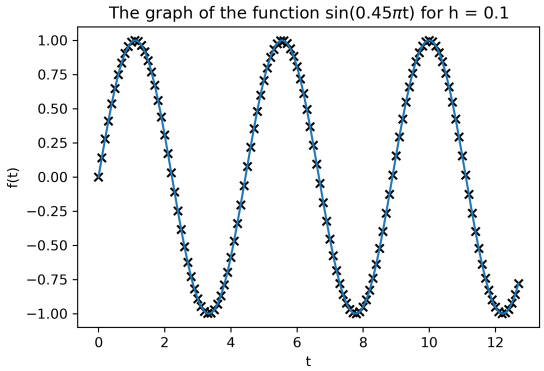


Figure 3.7: The graph of the function $f(t) = \sin(0.45\pi t)$ with the sampled points marked for $h = 0.1$

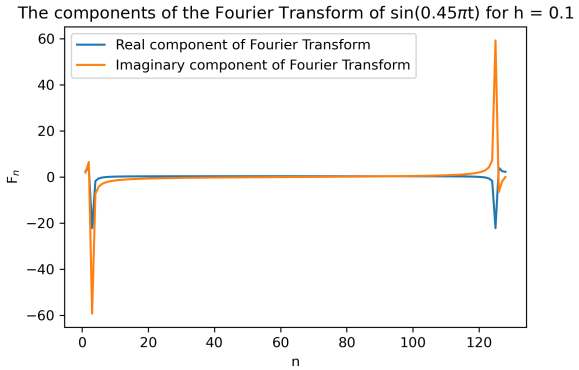


Figure 3.8: Fourier components of the function $f(t) = \sin(0.45\pi t)$ for $h = 0.1$

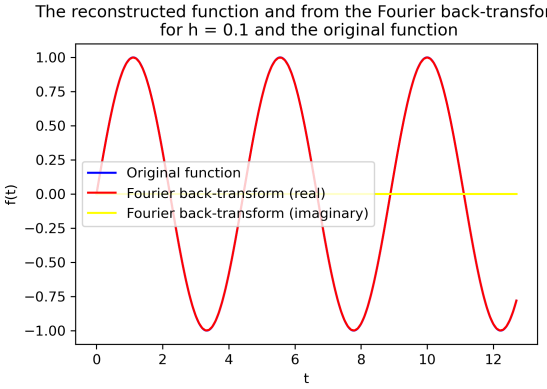


Figure 3.9: The discrete Fourier back-transform of the function $f(t) = \sin(0.45\pi t)$ for $h = 0.1$

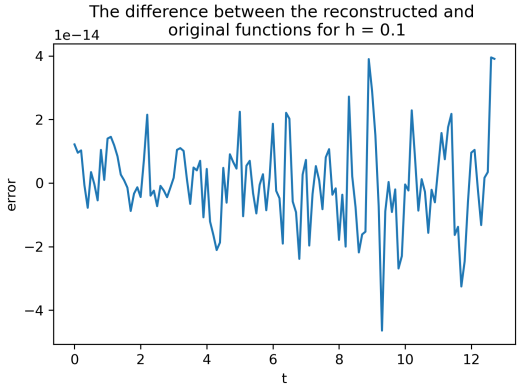


Figure 3.10: The error between the initial function and the Fourier back-transform of $f(t) = \sin(0.45\pi t)$ for $h = 0.1$

The discrete Fourier transform of the function $f(t) = \sin(0.45\pi t)$ was found with samples taken at a rate of 10 samples per second, giving the function a fundamental frequency $\omega_n = \frac{5\pi}{32} \approx 0.49087$. The graph in figure 3.7 shows the function and the individual sampled points. From this it can be expected that 63 ($\frac{N}{2} - 1 = \frac{64}{2} - 1$) individual frequencies can be calculated accurately.

The Fourier components are plotted in the graph in figure 3.8. The Nyquist symmetry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

From this the dominant component is seen to occur at $n = 3$ which is expected as when $0.45\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 2.88, whose closest integer is clearly 3. This produces a frequency for the function of $\frac{15\pi}{32} \approx 1.4726$, as opposed to the expected angular frequency of $\frac{9\pi}{20} \approx 1.414$.

The initial signal was also computed by the discrete Fourier back-transform and pro-

duced the graph in figure 3.9.

The real part of this graph is clearly very similar to the initial function, as shown both by visual inspection and as shown in the graph of the difference between the function and the back-transform of the function in figure 3.10 whose maximum is on the order of 10^{-14}. The imaginary part of this graph is, as expected, very small and essentially equal to zero as the initial function was real-valued only.

It was then found that an ideal value for h would be $\frac{5}{144}$ as it would allow for sampling over a single period, the above was repeated for this value of h.

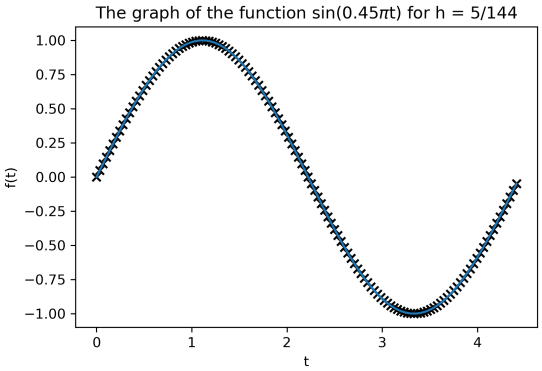


Figure 3.11: The graph of the function $f(t) = \sin(0.45\pi t)$ with the sampled points marked for $h = \frac{5}{144}$

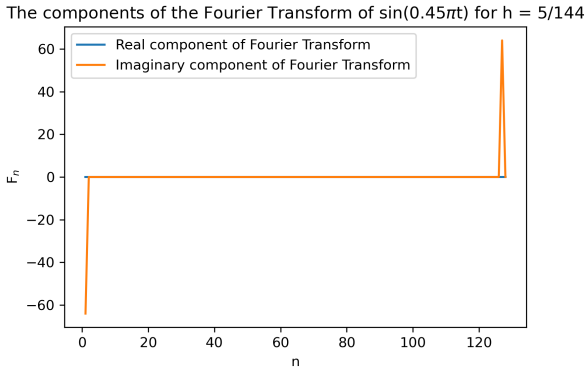


Figure 3.12: Fourier components of the function $f(t) = \sin(0.45\pi t)$ for $h = \frac{5}{144}$

The discrete Fourier transform of the function $f(t) = \sin(0.45\pi t)$ was found with samples taken at a rate of 28.8 samples per second, giving the function a fundamental frequency $\omega_n = 0.45\pi \approx 1.414$. The graph in figure 3.11 shows the function and the individual sampled points. From this it can be expected that 63 ($\frac{N}{2} - 1 = \frac{64}{2} - 1$) individual frequencies can be calculated accurately.

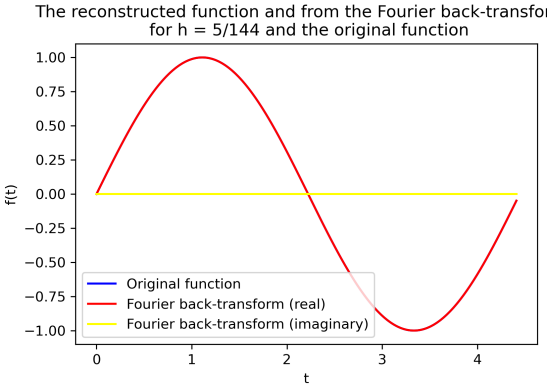


Figure 3.13: The discrete Fourier back-transform of the function $f(t) = \sin(0.45\pi t)$ for $h = \frac{5}{144}$

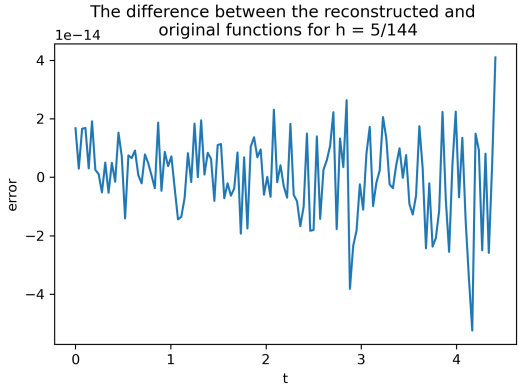


Figure 3.14: The error between the initial function and the Fourier back-transform of $f(t) = \sin(0.45\pi t)$ for $h = \frac{5}{144}$

The Fourier components are plotted in the graph in figure 3.12. The Nyquist symmetry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

From this the dominant component is seen to occur at $n = 1$ which is expected as when $0.45\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 1, whose closest integer is clearly 3. This produces a frequency for the function of 0.45π, which is in agreement with the expected angular frequency of $\frac{9\pi}{20}$.

The initial signal was also computed by the discrete Fourier back-transform and produced the graph in figure 3.13.

The real part is clearly very similar to the initial function, as shown both by visual inspection and as shown in the graph of the difference between the function and the back-transform of the function in figure 3.14 whose maximum is on the order of 10^{-14}. The imaginary part of this graph is, as expected, very small and essentially equal to zero

as the initial function was real-valued only.

3.3.2 $f(t) = cos(6\pi t)$

In all cases in this part $N = 32$ samples are taken and from this it can be expected that frequencies up to $F_{N/2-1} = F_{15}$ ($\frac{N}{2} - 1 = \frac{32}{2} - 1 = 15$) frequencies can be calculated accurately.

h=0.6

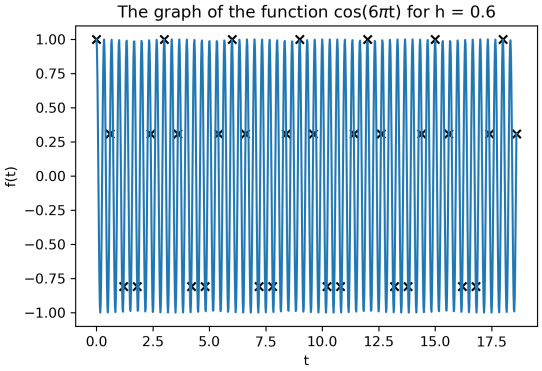


Figure 3.15: The graph of the function $f(t) = \sin(6\pi t)$ with the sampled points marked for $h = 0.6$

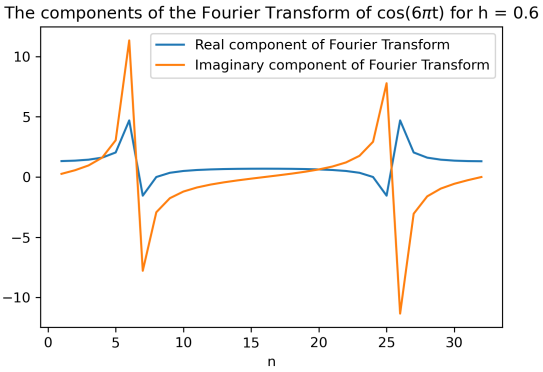


Figure 3.16: Fourier components of the function $f(t) = \sin(6\pi t)$ for $h = 0.6$

The discrete Fourier transform of the function $f(t) = \sin(6\pi t)$ was found with $N = 32$ samples taken at a rate of $1.\overline{6}$ samples per second, giving the function a fundamental frequency $\omega_n = \frac{5\pi}{48} \approx 0.3272$. The graph in figure 3.15 shows the function and the individual sampled points.

The Fourier components are plotted in the graph in figure 3.16. The Nyquist symmetry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the

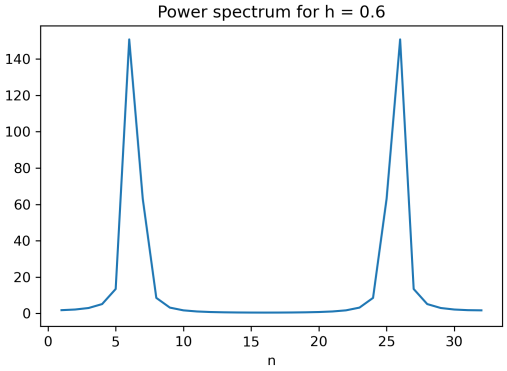


Figure 3.17: The power spectrum of the function $f(t) = \sin(6\pi t)$ for $h = 0.6$

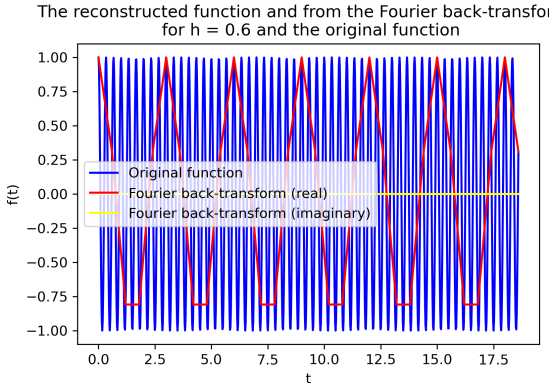


Figure 3.18: The discrete Fourier back-transform of the function $f(t) = \sin(6\pi t)$ for $h = 0.6$

real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

When $6\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 57.6 which is larger than the number of samples taken. This can lead us to conclude that enough samples were not taken from the signal, which is backed up by the graph of the real part of the Fourier back-transform in figure 3.18 of the reconstructed function from the Fourier back-transform. The real part of this reconstructed function while significantly different from the initial function does at least have the correct general shape of a vaguely sinusoidal function and the imaginary part of this graph is, as expected, very small and essentially equal to zero as the initial function was real-valued only. But as seen in figure 3.15 not enough samples are taken of the graph. This is also shown by the fact that the Nyquist frequency is $\nu_n = \frac{25}{32}$ which is less than the frequency of the function which is equal to 3. More frequent samples should be taken to gain an accurate description of this function.

Figure 3.17 shows the power spectrum for these conditions, with peaks at $n = 6$ and $n = 26$. This is what is expected as the value found for n above, 57.6 when rounded

to the nearest integer gives 58, which is equal to $32 + 26$. The power spectrum conveys the similar information to that of the graph of the components, the noticeable difference being that it combines the real and imaginary terms together, which makes it easier to identify the points at which the overall 'signal' is strongest but does cause some loss in information as to which component contributes the most to the signal.

h=0.2

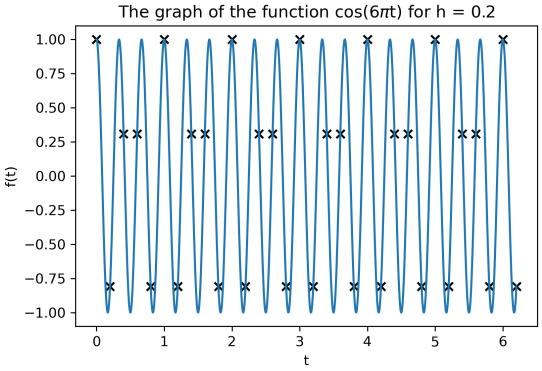


Figure 3.19: The graph of the function $f(t) = \sin(6\pi t)$ with the sampled points marked for $h = 0.2$

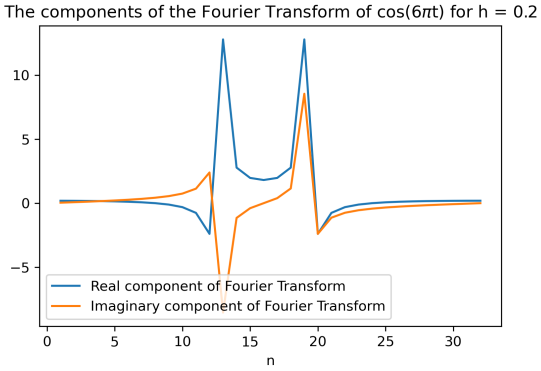


Figure 3.20: Fourier components of the function $f(t) = \sin(6\pi t)$ for $h = 0.2$

The discrete Fourier transform of the function $f(t) = \sin(6\pi t)$ was found with $N = 32$ samples taken at a rate of 5 samples per second, giving the function a fundamental frequency $\omega_n = \frac{5\pi}{16} \approx 0.9817$. The graph in figure 3.19 shows the function and the individual sampled points.

The Fourier components are plotted in the graph in figure 3.20. The Nyquist symmetry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

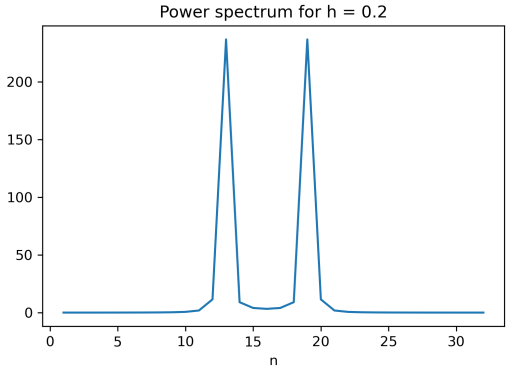


Figure 3.21: The power spectrum of the function $f(t) = \sin(6\pi t)$ for $h = 0.2$

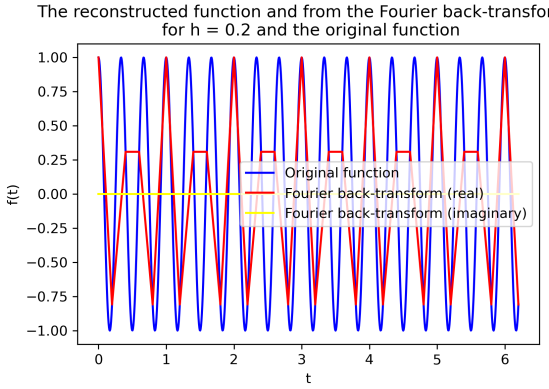


Figure 3.22: The discrete Fourier back-transform of the function $f(t) = \sin(6\pi t)$ for $h = 0.2$

From this the dominant component is seen to occur at $n = 19$ or $n = 13$ which is expected as when $6\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 19.2, which clearly rounds to 19. This is shown explicitly in figure 3.21 which shows the power spectrum for these conditions, with peaks at $n = 13$ and $n = 19$.

The Nyquist frequency is $\nu_n = \frac{75}{32}$ which is less than the frequency of the function which is equal to 3. This suggests that not enough samples are taken from the signal, which is backed up by the graph of the real part of the Fourier back-transform in figure 3.22 of the reconstructed function from the Fourier back-transform. The real part of this reconstructed function while significantly different from the initial function does at least have the correct general shape of a vaguely sinusoidal function and the imaginary part of this graph is, as expected, very small and essentially equal to zero as the initial function was real-valued only. But as seen in figure 3.19 not enough samples are taken of the graph. More frequent samples should be taken to gain an accurate description of this function.

h=0.1

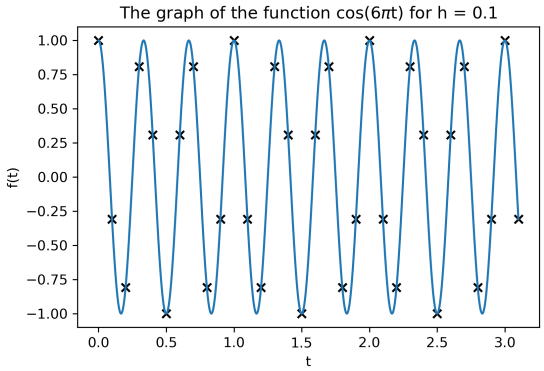


Figure 3.23: The graph of the function $f(t) = \sin(6\pi t)$ with the sampled points marked for $h = 0.1$

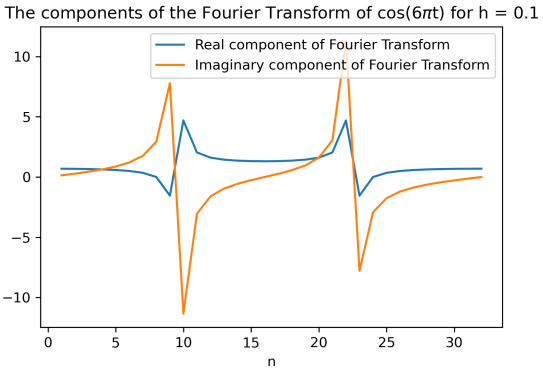


Figure 3.24: Fourier components of the function $f(t) = \sin(6\pi t)$ for $h = 0.1$

The discrete Fourier transform of the function $f(t) = \sin(6\pi t)$ was found with $N = 32$ samples taken at a rate of 10 samples per second, giving the function a fundamental frequency $\omega_n = \frac{5\pi}{8} \approx 1.963$. The graph in figure 3.23 shows the function and the individual sampled points.

The Fourier components are plotted in the graph in figure 3.24. The Nyquist symmetry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

From this the dominant component is seen to occur at $n = 10$ or $n = 22$ which is expected as when $6\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 9.6, which clearly rounds to 10. This is shown explicitly in figure 3.21 which shows the power spectrum for these conditions, with peaks at $n = 10$ and $n = 22$.

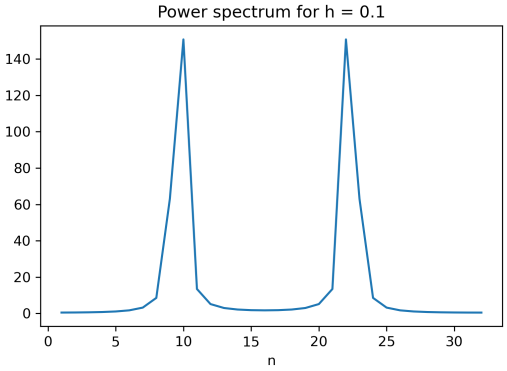


Figure 3.25: The power spectrum of the function $f(t) = \sin(6\pi t)$ for $h = 0.1$

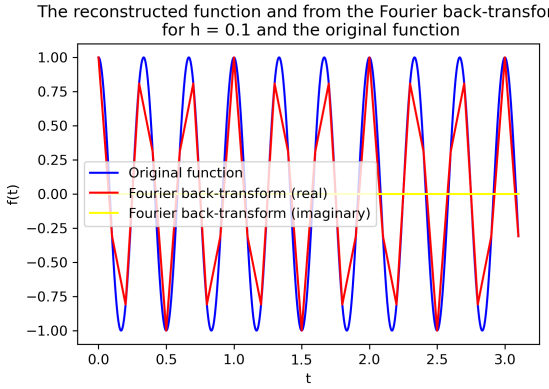


Figure 3.26: The discrete Fourier back-transform of the function $f(t) = \sin(6\pi t)$ for $h = 0.1$

The Nyquist frequency is $\nu_n = \frac{75}{16}$ which is greater than the frequency of the function which is equal to 3. This suggests that enough samples are taken from the signal, to get an accurate Fourier transform of it. This is backed up by the graph of the real part of the Fourier back-transform in figure 3.26 of the reconstructed function from the Fourier back-transform. This reconstructed function is very similar to the original function. It is much sharper and more pointed than the original, however this is due to a limitation in the number of samples and with more samples it should produce a smoother graph. The imaginary part of this graph is, as expected, very small and essentially equal to zero as the initial function was real-valued only.

h=0.04

The discrete Fourier transform of the function $f(t) = \sin(6\pi t)$ was found with $N = 32$ samples taken at a rate of 25 samples per second, giving the function a fundamental frequency $\omega_n = \frac{25\pi}{16} \approx 4.909$. The graph in figure 3.27 shows the function and the individual sampled points.

The Fourier components are plotted in the graph in figure 3.28. The Nyquist sym-

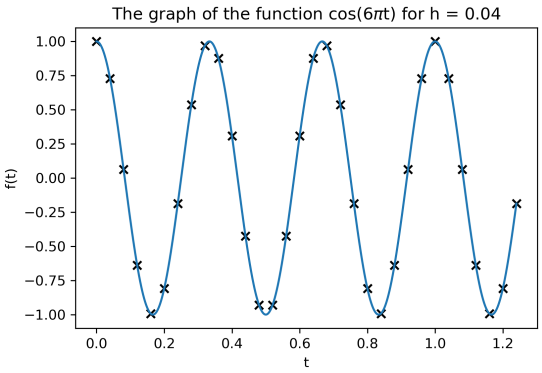


Figure 3.27: The graph of the function $f(t) = \sin(6\pi t)$ with the sampled points marked for $h = 0.04$

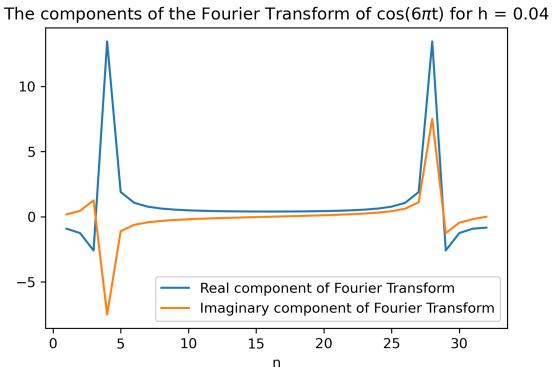


Figure 3.28: Fourier components of the function $f(t) = \sin(6\pi t)$ for $h = 0.04$

metry, $F_{N-n} = F_n^*$ is readily visible in the graph from the fact that the peaks for the real components are symmetrical about the centre line of the graph while the imaginary components are symmetric about the centre point of the graph.

From this the dominant component is seen to occur at $n = 4$ or $n = 28$ which is expected, as when $6\pi t_m = \frac{2\pi nm}{N}$ is solved for n it is found to equal 3.84, which clearly rounds to 4. This is shown explicitly in figure 3.29 which shows the power spectrum for these conditions, with peaks at $n = 4$ and $n = 28$.

The Nyquist frequency is $\nu_n = \frac{375}{32}$ which is greater than the frequency of the function which is equal to 3. This suggests that enough samples are taken from the signal, to get an accurate Fourier transform of it. This is backed up by the graph of the real part of the Fourier back-transform in figure 3.30 of the reconstructed function from the Fourier back-transform. This reconstructed function is very similar to the original function. It is much sharper and more pointed than the original, however this is due to a limitation in the number of samples and with more samples it should produce a smoother graph, indeed it is smoother than the graph in figure 3.26 which was generated from samples

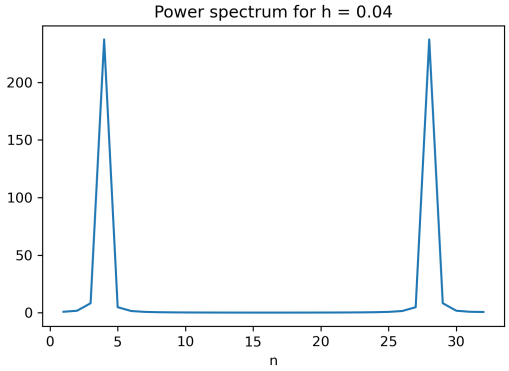


Figure 3.29: The power spectrum of the function $f(t) = \sin(6\pi t)$ for $h = 0.04$

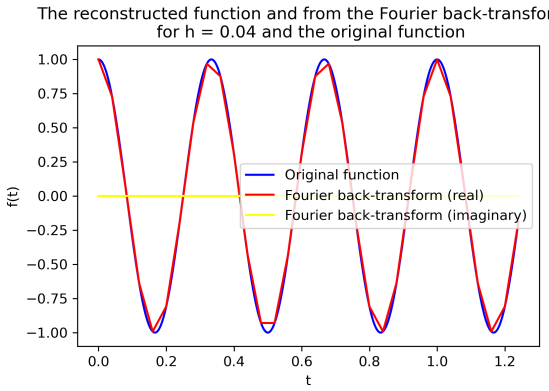


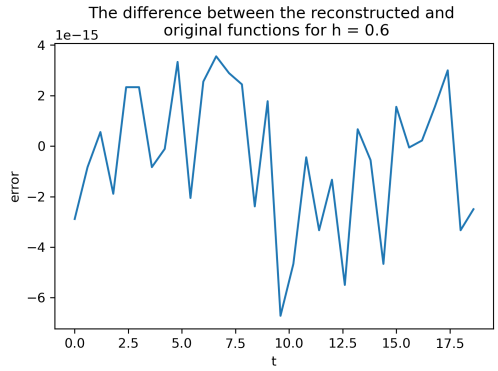
Figure 3.30: The discrete Fourier back-transform of the function $f(t) = \sin(6\pi t)$ for $h = 0.04$

taken at a rate of 10 times per second. The imaginary part of this graph is, as expected, very small and essentially equal to zero as the initial function was real-valued only.

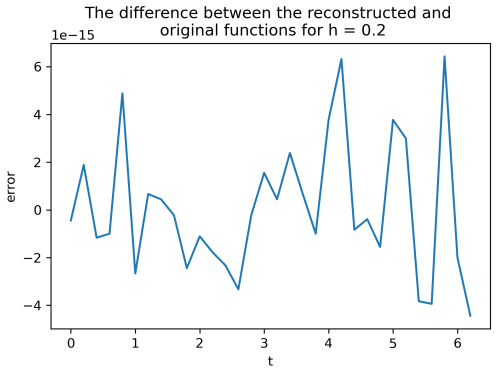
Error plots

The plots of the difference between the original function and the back-transform are shown in figure 3.31. Their low values should not be taken at face-value. As this is a discrete Fourier back-transform only the difference between some discrete, sampled points and their values after being Fourier transformed and subsequently back-transformed.

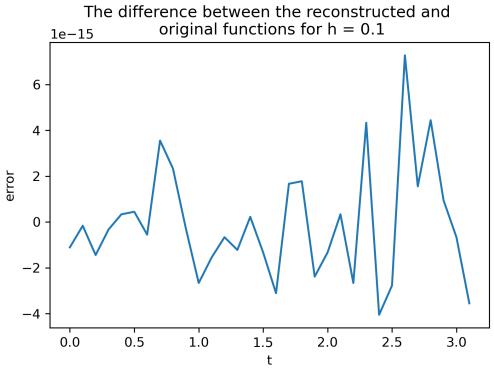
This is why, despite the (in some cases large) difference between much of the real values of the back-transformed graph and the initial function the error values are calculated to be really low, because the sampled points are reproduced almost exactly in the positions they were.



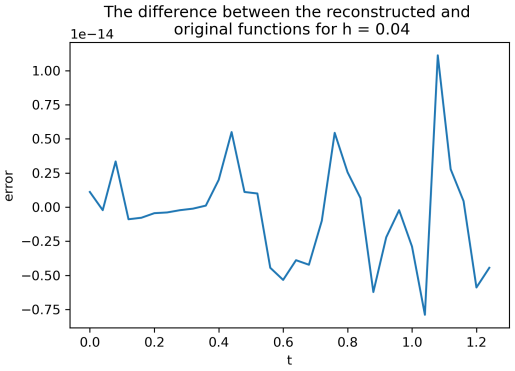
(a) The error between the initial function and the Fourier back-transform for $h = 0.6$.



(b) The error between the initial function and the Fourier back-transform for $h = 0.2$.



(c) The error between the initial function and the Fourier back-transform for $h = 0.1$.



(d) The error between the initial function and the Fourier back-transform for $h = 0.04$.

Figure 3.31: The error between the initial function and the Fourier back-transform of the function $f(t) = \sin(6\pi t)$ for varying values of h.

4 Conclusions

Simpson's Rule is a very good approximation for integration as shown by its ability to accurately calculate the integral $\int_0^1 e^x dx$ as well as the Fourier coefficients of a number of both sinusoidal and non-sinusoidal periodic functions. As the number of Fourier coefficients used is increased a more accurate Fourier series for any given signal is generated.

The discrete Fourier transform of a function can be used to find its frequency, and as more frequent samples are taken it becomes a more accurate representation of the original function.

Python is a useful tool for Fourier analysis as it is possible to carry out many similar, repetitive tasks easily with it, allowing for short computation times and fast calculation of the many quantities which must be found when carrying out Fourier analysis.

5 Appendices

5.1 Appendix A: Derivations

5.1.1 Derivation of Fourier Coefficients for the Square Wave

From Eq. 1.4 above, and $f(t)$ as defined in Eq. 2.1

$$b_n = \frac{1}{\pi}\left(\int_0^{\pi} \sin(nt)dt - \int_{\pi}^{2\pi} \sin(nt)dt\right)$$

$$= \frac{1}{\pi n}\left(-(\cos(nt))|_0^{\pi} + \cos(nt)|_{\pi}^{2\pi}\right)$$

$$= \frac{1}{\pi n}\left(-(\cos(\pi n) - 1) + (1 - \cos(\pi n))\right)$$

$$= \frac{2(1 - cos(\pi n))}{\pi n}$$

$$= \frac{2(1 - (-1)^n}{\pi n}$$

Which for even n $b_n = 0$. For odd n,

$$b_n = \frac{2(1+1)}{\pi n} = \frac{4}{\pi n} \tag{5.1}$$

5.1.2 Derivation of Fourier Coefficients for the Rectangular Wave

In all cases here $f(t)$ is as defined in Eq. 2.3

From Eq. 1.2 above a_0 is calculated as follows,

$$a_0 = \frac{1}{2\pi}\left(\int_0^{1} 1dt - \int_{1}^{2\pi} 1dt\right)$$

$$= \frac{1}{2\pi}\left(t|_0^{1} - t|_1^{2\pi}\right)$$

$$= \frac{1}{2\pi}\left((1 - 0) - (2\pi - 1)\right)$$

$$= \frac{1}{2\pi}\left(2 - 2\pi\right)$$

$$a_0 = \frac{1 - \pi}{\pi} \approx -0.6817 \tag{5.2}$$

From Eq. 1.3 above a_n is calculated as follows,

$$a_n = \frac{1}{\pi}\left(\int_0^1 \cos(nt)dt - \int_1^{2\pi} \cos(nt)dt\right)$$

$$= \frac{1}{\pi n}\left(\sin(nt)|_0^1 - \sin(nt)|_1^{2\pi}\right)$$

$$= \frac{1}{\pi n}\left((\sin(n) - 0) - (0 - \sin(n))\right)$$

$$= \frac{1}{\pi n}\left(\sin(n) + \sin(n)\right)$$

$$a_n = \frac{2\sin(n)}{\pi n} \tag{5.3}$$

From Eq. 1.4 above b_n is calculated as follows,

$$b_n = \frac{1}{\pi}\left(\int_0^1 \sin(nt)dt - \int_1^{2\pi} \sin(nt)dt\right)$$

$$= \frac{1}{\pi n}\left(-\cos(nt)|_0^1 + \cos(nt)|_1^{2\pi}\right)$$

$$= \frac{1}{\pi n}\left(-(\cos(n) - 1) + (1 - \cos(n))\right)$$

$$= \frac{1}{\pi n}\left(-\cos(n) + 1 + 1 - \cos(n)\right)$$

$$b_n = \frac{2 - 2\cos(n)}{\pi n} \tag{5.4}$$

5.1.3 Nyquist Symmetry

The Nyquist symmetry states that $F_{N-n} = F_n^*$.

$$F_{N-n} = \sum_{m=0}^{N-1} f_m \cos\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) - f_m \sin\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) \tag{5.5}$$

And

$$F*_n = \sum_{m=0}^{N-1} f_m \cos\left(\frac{2\pi imn}{N}\right) + f_m \sin\left(\frac{2\pi imn}{N}\right) \tag{5.6}$$

As the summation is the same the terms inside it must be equal,

$$f_m \cos\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) - f_m \sin\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) = f_m \cos\left(\frac{2\pi imn}{N}\right) + f_m \sin\left(\frac{2\pi imn}{N}\right)$$

$$\Rightarrow \cos\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) - \sin\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) = \cos\left(\frac{2\pi imn}{N}\right) + \sin\left(\frac{2\pi imn}{N}\right) \tag{5.7}$$

Taking the left-hand side of this expression,

$$\begin{aligned}
&\cos\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) - \sin\left(\frac{2\pi imN}{N} - \frac{2\pi imn}{N}\right) \\
&= \cos\left(\frac{2\pi mN}{N}\right)\cos\left(\frac{2\pi mn}{N}\right) + \sin\left(\frac{2\pi mN}{N}\right)\sin\left(\frac{2\pi mn}{N}\right) \\
&- \sin\left(\frac{2\pi mN}{N}\right)\cos\left(\frac{2\pi mn}{N}\right) + \cos\left(\frac{2\pi mN}{N}\right)\sin\left(\frac{2\pi mn}{N}\right) \\
&= \cos\left(\frac{2\pi imn}{N}\right) + \sin\left(\frac{2\pi imn}{N}\right)
\end{aligned} \tag{5.8}$$

As $\cos\left(\frac{2\pi mN}{N}\right) = \cos(2\pi m) = 1$ for all $m \in \mathbb{Z}$ and $\sin\left(\frac{2\pi mN}{N}\right) = \sin(2\pi m) = 0$ for all $m \in \mathbb{Z}$.

This is equal the right-hand side of Eq. 5.7 and so,

$$F_{N-n} = F_n^* \tag{5.9}$$

5.2 Appendix B: Code

5.2.1 Using Simpson's Rule to Evaluate Fourier Series

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 28 14:05:30 2023

@author: wattersb
"""
import numpy as np
import matplotlib.pyplot as plt

#####################################
#                                   #
#   SIMPSONS RULE FOR INTEGRATION   #
#                                   #
#####################################


```

```
def Simpson(f, a, b, n): # FUNCTION, LOWER LIMIT, UPPER LIMIT, NUMBER OF STEPS
    n = int(n)*2 # ENSURE ONLY EVEN NUMBERS OF STEPS ARE USED
    h = (b-a) / n

    # THE SUMMATION TERMS
    sigma1 = 0
    sigma2 = 0

    for j in range(1, int(n/2)): # LOOP OVER [1,n/2 - 1]
        x2j = a + 2*j*h
        sigma1 += f(x2j)

    for j in range(1, int(n/2 + 1)): # LOOP OVER [1,n/2]
        xj = a + (2*j-1)*h
        sigma2 += f(xj)

    return h/3 * (f(a) + 2*sigma1 + 4*sigma2 + f(b))

print("--- Testing the Simpson's Rule code ---")

print("Integral evaluated using Simpson's Rule "+str(Simpson(np.exp, 0, 1, 4)))

print("Difference between Simpson's Rule and actual value "+str(Simpson(np.exp, 0, 1, 4) - (np.exp(1) - np.exp(0))))


##############################################
#                                            #
#   DEFINE FUNCTIONS TO FIND COEFFICIENTS    #
#                                            #
##############################################


def a0(f, omega, n): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS/2
    T = 2*np.pi/omega
    return 1/T * Simpson(f, 0, T, n)

def ak(f, omega, n, k): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS/2, k STEPS
    T = 2*np.pi/omega

    # EMPTY ARRAY OF ak VALUES
    ak = np.array([])

    # THE FUNCTION TIMES THE COSINE TERM
    def fcos(t):
        return f(t)*np.cos(t*omega*i)

    # CALCULATE THE ak TERMS
    for i in range(1, k+1):
```

```
65          a = Simpson(fcos, 0, T, n)
66
67          ak = np.append(ak, a)
68
69      ak = 2/T * ak
70      return ak
71
72  def bk(f, omega, n, k): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS
        /2, k STEPS
73      T = 2*np.pi/omega
74
75      # EMPTY ARRAY OF bk VALUES
76      bk = np.array([])
77
78      # THE FUNCTION TIMES THE SINE TERM
79      def fsin(t):
80          return f(t)*np.sin(t*omega*i)
81
82      # CALCULATE THE bk TERMS
83      for i in range(1, k+1):
84          b = Simpson(fsin, 0, T, n)
85
86          bk = np.append(bk, b)
87
88      bk = 2/T * bk
89      return bk
90
91  def fourierSeries(f, omega, n, k, t):
92
93      # CALCULATE THE COEFFICIENTS
94      azero = a0(f, omega, n)
95      a = ak(f, omega, n, k)
96      b = bk(f, omega, n, k)
97
98      # CALCULATE THE SERIES
99      fourier = 0
100     for m in range(1, len(a)+1):
101         fourier += a[m-1]*np.cos(m*t) + b[m-1]*np.sin(m*t)
102     fourier += azero
103
104     return azero, a, b, fourier
105
106
107 ################################
108 #                              #
109 #   FIND THE FOURIER SERIES    #
110 #                              #
111 ################################
112
113
114 # FUNCTION TO GRAPH THE FUNCTION, SERIES AND COEFFICIENTS
115 def graphs(t_list, function, fourier, K, azero, a, b, funcLabel):
```

```
116     k_list = np.arange(1,K+1,1)
117 
118     # FUNCTION
119     plt.title("Graph of f(t) and its fourier series")
120     plt.plot(t_list, function(t_list), label="f(t) = " + str(funcLabel)
    )
121     plt.plot(t_list, fourier, label="Fourier series of f(t) = " + str(
    funcLabel))
122     plt.legend(loc=1)
123     plt.xlabel("t")
124     plt.ylabel("f(t)")
125     plt.savefig("images/exercise1_series_" + str(funcLabel) + ".png",
    dpi=300)
126     plt.show()
127 
128     # COEFFICIENTS
129     plt.title("The coefficients of the fourier series of \nf(t) = " +
    str(funcLabel))
130     plt.scatter(0, azero, color='blue', s=20, label="a$_n$, n $\geq$ 0"
    , marker="x")
131     plt.scatter(k_list, a, color='blue', s=20, marker="x")
132     plt.scatter(k_list, b, color='red', label="b$_n$, n $\geq$ 1",
    marker="+")
133     plt.legend(loc=1)
134     plt.xlim([-.5,K+0.5])
135     plt.xlabel("n")
136     plt.ylabel("a$_n$, b$_n$")
137     plt.savefig("images/exercise1_coefficients_" + str(funcLabel) + ".
    png", dpi=300)
138     plt.show()
139 
140 def coefficients(K, azero, a, b, funcLabel):
141     print("--- Fourier Series coefficients for f(t) = " + str(funcLabel
    ) + "---")
142     print("a0 = "+str(azero))
143 
144     for i in range(1, K+1):
145         print("a"+str(i)+" = "+str(a[i-1]))
146     print("-")
147     for i in range(1, K+1):
148         print("b"+str(i)+" = "+str(b[i-1]))
149 
150 t_list = np.linspace(0,2*np.pi,1000)
151 K = 10
152 
153 ##################
154 # f(t) = sin(t) #
155 ##################
156 
157 # DEFINE FUNCTION
158 def function(t):
159     return np.sin(t)
```

```
160
161 # GET VALUES
162 azero, a, b, fourier = fourierSeries(function, 1, 15, K, t_list)
163
164 # GRAPHS
165 graphs(t_list, function, fourier, K, azero, a, b, "sin(t)")
166
167 # COEFFICIENTS
168 coefficients(K, azero, a, b, "sin(t)")
169
170 ########################################
171 # f(t) = cos(t) + 3cos(2t) - 4cos(3t) #
172 ########################################
173
174 # DEFINE FUNCTION
175 def function(t):
176     return np.cos(t) + 3*np.cos(2*t) - 4*np.cos(3*t)
177
178 # GET VALUES
179 azero, a, b, fourier = fourierSeries(function, 1, 15, K, t_list)
180
181 # GRAPHS #
182 graphs(t_list, function, fourier, K, azero, a, b, "cos(t) + 3cos(2t) -
    4cos(3t)")
183
184 # COEFFICIENTS
185 coefficients(K, azero, a, b, "cos(t) + 3cos(2t) - 4cos(3t)")
186
187 ########################################
188 # f(t) = sin(t) + 3sin(3t) + 5sin(5t) #
189 ########################################
190
191 # DEFINE FUNCTION
192 def function(t):
193     return np.sin(t) + 3*np.sin(3*t) + 5*np.sin(5*t)
194
195 # GET VALUES
196 azero, a, b, fourier = fourierSeries(function, 1, 20, K, t_list)
197
198 # GRAPHS #
199 graphs(t_list, function, fourier, K, azero, a, b, "sin(t) + 3sin(3t) +
    5sin(5t)")
200
201 # COEFFICIENTS
202 coefficients(K, azero, a, b, "sin(t) + 3sin(3t) + 5sin(5t)")
203
204 ########################################
205 # f(t) = sin(t) + 2cos(3t) + 3sin(5t) #
206 ########################################
207
208 # DEFINE FUNCTION
209 def function(t):
```

```
    return np.sin(t) + 2*np.cos(3*t) + 3*np.sin(5*t)

# GET VALUES
azero, a, b, fourier = fourierSeries(function, 1, 20, K, t_list)

# GRAPHS #
graphs(t_list, function, fourier, K, azero, a, b, "sin(t) + 2cos(3t) +
    3sin(5t)")

# COEFFICIENTS
coefficients(K, azero, a, b, "sin(t) + 2cos(3t) + 3sin(5t)")
```

5.2.2 Finding the Fourier Series of Square and Rectangular Waves

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  4 14:13:48 2023

@author: wattersb
"""
import numpy as np
import matplotlib.pyplot as plt

# FUNDANMENTAL FREQUENCY
omega = 1

# TAU FOR RECTANGULAR WAVE
tau = 1

# SIMPSONS RULE FOR INTEGRATION
def Simpson(f, a, b, n): # FUNCTION, LOWER LIMIT, UPPER LIMIT, NUMBER
    OF STEPS

    h = (b-a) / n

    # THE SUMMATION TERMS
    sigma1 = 0
    sigma2 = 0

    j = 1
    while j <= n/2 - 1:
        x2j = a + 2*j*h
        sigma1 = sigma1 + f(x2j)

        j += 1

    j = 1
    while j <= n/2:
        xj = a + (2*j-1)*h
        sigma2 = sigma2 + f(xj)
        j += 1

    return h/3 * (f(a) + 2*sigma1 + 4*sigma2 + f(b))
```

```
39
40
41 ##############################################
42 #                                            #
43 #   DEFINE FUNCTIONS TO FIND COEFFICIENTS    #
44 #                                            #
45 ##############################################
46
47
48 def a0(f, omega, n): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS
49     T = 2*np.pi/omega
50     return 1/T * Simpson(f, 0, T, n)
51
52 def ak(f, omega, n, k): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS,
       k STEPS
53     T = 2*np.pi/omega
54
55     # EMPTY ARRAY OF ak VALUES
56     ak = np.array([])
57
58     # THE FUNCTION TIMES THE COSINE TERM
59     def fcos(x):
60         return f(x)*np.cos(x*omega*i)
61
62     # CALCULATE THE ak TERMS
63     for i in range(1, k+1):
64         a = Simpson(fcos, 0, T, n)
65         ak = np.append(ak, a)
66
67     ak = 2/T * ak
68     return ak
69
70 def bk(f, omega, n, k): # NATURAL FREQUENCY, FUNCTION, NUMBER OF STEPS,
       k STEPS
71     T = 2*np.pi/omega
72
73     # EMPTY ARRAY OF bk VALUES
74     bk = np.array([])
75
76     # THE FUNCTION TIMES THE SINE TERM
77     def fsin(x):
78         return f(x)*np.sin(x*omega*i)
79
80     # CALCULATE THE bk TERMS
81     for i in range(1, k+1):
82         b = Simpson(fsin, 0, T, n)
83         bk = np.append(bk, b)
84
85     bk = 2/T * bk
86     return bk
87
88 def fourierSeries(f, omega, n, k, t): # NATURAL FREQUENCY, FUNCTION,
```

```
    NUMBER OF STEPS, k STEPS, t VALUE

    # CALCULATE THE COEFFICIENTS
    azero = a0(f, omega, n)
    a = ak(f, omega, n, k)
    b = bk(f, omega, n, k)

    # CALCULATE THE SERIES
    fourier = 0
    for n in range(1, k+1):
        fourier += a[n-1]*np.cos(n*t) + b[n-1]*np.sin(n*t)
    fourier += azero

    return azero, a, b, fourier


####################
#                  #
#   SQUARE WAVE    #
#                  #
####################


def square(t_list):

    step_list = np.array([])

    # USED TO ENSURE THE FUNCTION IS PLOTTED CORRECTLY OVER ALL RANGES
    def squareFunc(t_list):
        theta = t_list * omega

        # KEEP THETA WITHIN [0, 2pi]
        while np.abs(theta) > 2*np.pi:
             theta = theta - 2*np.pi * np.abs(theta)/theta

        # ASSIGN VALUES
        if (theta >= 0 and theta <= np.pi) or (theta < -np.pi and theta
    >= -2*np.pi):
            return 1
        else:
            return -1

    # WORKS SLIGHTLY DIFFERENTLY FOR NUMBERS AND ARRAYS - PYTHON WON'T
    "LOOP" OVER NUMBERS
    if type(t_list) == float or type(t_list) == int:
        step_list = np.append(step_list, squareFunc(t_list))
    else:
        for t in t_list:
            step_list = np.append(step_list, squareFunc(t))

    return step_list

```

```
138
139 # GET VALUES
140 t_list = np.linspace(0,2*np.pi,2000)
141 k_values = np.array([1,2,3,5,10,20,30])
142 azero, a, b, fourier = fourierSeries(square, 1, 1000, k_values[-1],
        t_list)
143
144 ##########
145 # GRAPHS #
146 ##########
147
148 # TOGETHER
149 plt.title("Graph of the square wave with its Fourier series for\
        nvarious values of n superimposed")
150 plt.plot(t_list, square(t_list), label="Square function")
151 for i in k_values:
152     plt.plot(t_list, fourierSeries(square, 1, 1000, i, t_list)[3],
        label="n=" + str(i))
153
154 plt.legend(loc=1)
155 plt.xlabel("t")
156 plt.ylabel("f(t)")
157 plt.savefig("images/exercise2_square.png", dpi=300)
158 plt.show()
159
160 # DIFFERENT K VALUES
161 for i in k_values:
162     plt.title("Graph of the square wave with its Fourier series for n =
        "+str(i))
163     plt.plot(t_list, square(t_list), label="Square function")
164     plt.plot(t_list, fourierSeries(square, 1, 1000, i, t_list)[3],
        label="Fourier series with n=" + str(i)  )
165     plt.legend(loc=1)
166     plt.xlabel("t")
167     plt.ylabel("f(t)")
168     plt.savefig("images/exercise2_square_k" + str(i) + ".png", dpi=300)
169     plt.show()
170
171 # OUTSIDE OF [0, 2pi], k = 30
172 t_list = np.linspace(-10,10,5000)
173
174 plt.title("Graph of the square wave with its Fourier series for n = 30"
        )
175 plt.plot(t_list, square(t_list), label="Square function")
176 plt.plot(t_list, fourierSeries(square, 1, 1000, 30, t_list)[3], label="
        Fourier series with n=30")
177 plt.legend(loc=1)
178 plt.xlabel("t")
179 plt.ylabel("f(t)")
180 plt.savefig("images/exercise2_square_long.png", dpi=300)
181 plt.show()
182
```

```
# PRINT COEFFICIENTS
print("--- Fourier Series coefficients for the square wave function ---")
print("a0 = "+str(azero))

for i in range(1, k_values[-1]+1):
    print("a"+str(i)+" = "+str(a[i-1]))
print("-")
for i in range(1, k_values[-1]+1):
    print("b"+str(i)+" = "+str(b[i-1]))


########################
#                      #
#   RECTANGULAR WAVE   #
#                      #
########################


def rectangular(t_list):

    # USED TO ENSURE THE FUNCTION IS PLOTTED CORRECTLY OVER ALL RANGES
    def rectangularFunc(t_list):
        theta = t_list * omega

        # KEEP THETA WITHIN [0, 2pi]
        while np.abs(theta) > 2*np.pi:
             theta = theta - 2*np.pi * np.abs(theta)/theta

        # ASSIGN VALUES
        if (theta >= 0 and theta <= omega * tau) or (theta >= -2*np.pi and theta <= -2*np.pi + omega * tau):
            return 1
        else:
            return -1

    step_list = np.array([])

    # WORKS SLIGHTLY DIFFERENTLY FOR NUMBERS AND ARRAYS - PYTHON WON'T "LOOP" OVER NUMBERS
    if type(t_list) == float or type(t_list) == int:
        step_list = np.append(step_list, rectangularFunc(t_list))
    else:
        for t in t_list:
            while np.abs(t) > 2* np.pi:
                t = t - 2*np.pi * np.abs(t)/t
            step_list = np.append(step_list, rectangularFunc(t))

    return step_list


# GET VALUES
```

```
t_list = np.linspace(0,2*np.pi,2000)
k_values = np.array([1,2,3,5,10,20,30])
azero, a, b, fourier = fourierSeries(rectangular, 1, 1000, k_values
    [-1], t_list)

##########
# GRAPHS #
##########

# TOGETHER
plt.title("Graph of the rectangular wave with its Fourier series for\
    nvarious values of n superimposed")
plt.plot(t_list, rectangular(t_list), label="Rectangular function")
for i in k_values:
    plt.plot(t_list, fourierSeries(rectangular, 1, 1000, i, t_list)[3],
    label="n=" + str(i))

plt.legend(loc=1)
plt.xlabel("t")
plt.ylabel("f(t)")
plt.savefig("images/exercise2_rect.png", dpi=300)
plt.show()

# DIFFERENT K VALUES
for i in k_values:
    plt.title("Graph of the rectangular wave with its Fourier series
    for n = "+str(i))
    plt.plot(t_list, rectangular(t_list), label="Rectangular function")
    plt.plot(t_list, fourierSeries(rectangular, 1, 1000, i, t_list)[3],
    label="Fourier series with n=" + str(i))
    plt.legend(loc=1)
    plt.xlabel("t")
    plt.ylabel("f(t)")
    plt.savefig("images/exercise2_rect_k" + str(i) + ".png", dpi=300)
    plt.show()

# OUTSIDE OF [0, 2pi], k = 30
t_list = np.linspace(-10,10,5000)

plt.title("Graph of the rectangular wave with its Fourier series for n
    = 30")
plt.plot(t_list, rectangular(t_list), label="Rectangular function")
plt.plot(t_list, fourierSeries(rectangular, 1, 1000, 30, t_list)[3],
    label="Fourier series with n=30")
plt.legend(loc=1)
plt.xlabel("t")
plt.ylabel("f(t)")
plt.savefig("images/exercise2_rect_long.png", dpi=300)
plt.show()

# PRINT COEFFICIENTS
print("--- Fourier Series coefficients for the rectangular wave
```

```
   function ---")
print("a0 = "+str(azero))

for i in range(1, k_values[-1]+1):
    print("a"+str(i)+" = "+str(a[i-1]))
print("-")
for i in range(1, k_values[-1]+1):
    print("b"+str(i)+" = "+str(b[i-1]))
```

5.2.3 The Discrete Fourier Transform in Python

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr  5 11:39:47 2023

@author: wattersb
"""
import numpy as np
import matplotlib.pyplot as plt


#########################
#                       #
#   DEFINE FUNCTIONS    #
#                       #
#########################


######################
# FOURIER TRANSFORM #
######################

# REAL PART
def fReal(f, N, n, h):

    sigma = 0
    for m in range(N):
        sigma += f(m*h) * np.cos(2*np.pi*m*n/N)

    return sigma

# IMAGINARY PART
def fImag(f, N, n, h):

    sigma = 0
    for m in range(N):
        sigma += -f(m*h) * np.sin(2*np.pi*m*n/N)

    return sigma


##################
# BACK TRANSFORM #
```

```
##################

def fmReal(Freal, Fimag, N, m):
    sigma = 0

    for n in range(N):
        sigma += Freal(n) * np.cos(2*np.pi*m*n/N) - Fimag(n) * np.sin
    (2*np.pi*m*n/N)

    return 1/N * sigma

def fmImag(Freal, Fimag, N, m):
    sigma = 0

    for n in range(N):
        sigma += Freal(n) * np.sin(2*np.pi*m*n/N) + Fimag(n) * np.cos
    (2*np.pi*m*n/N)

    return 1/N * sigma


####################
#                  #
#   sin(0.45pit)   #
#                  #
####################


# FUNCTION BEING ANALYSED
def func(t):
    return np.sin(0.45*np.pi*t)

##############
# TRANSFORM #
##############

N = 128 # number of samples
h_list = [0.1, 5/144] # sampling interval
h_labels = ["0.1", "5/144"]

x = 0
while x < len(h_list):

    h = h_list[x]

    # SAMPLING RATE
    rate = 1/h
    print("The sampling rate is " + str(rate) + " samples per second")

    # FUNDAMENTAL FREQUENCY
    omega1 = 2*np.pi/(h*N)
    print("The fundamental frequency is " + str(omega1) + " /s")
```

```
93
94      # PLOT THE FUNCTION
95      t_list = np.linspace(0, (N-1) * h, 1000)
96      plt.plot(t_list, func(t_list))
97
98      # SAMPLE TIMES
99      m = np.arange(0, N, 1)
100     tm = m*h
101
102     # ADD SAMPLED POINTS
103     plt.scatter(tm, func(tm), marker='x', color='black')
104     plt.xlabel("t")
105     plt.ylabel("f(t)")
106     plt.title("The graph of the function sin(0.45$\pi$t) for h = " +
    h_labels[x])
107     plt.savefig("images/exercise3_0.45_graph_"+str(h)+".png", dpi=300)
108     plt.show()
109
110     real = np.array([])
111     imag = np.array([])
112
113     n = np.arange(1, N+1, 1)
114     for i in n:
115         real = np.append(real, fReal(func, N, i, h))
116         imag = np.append(imag, fImag(func, N, i, h))
117
118     plt.title("The components of the Fourier Transform of sin(0.45$\
    pi$t) for h = " + h_labels[x])
119     plt.plot(n, real, label="Real component of Fourier Transform")
120     plt.plot(n, imag, label="Imaginary component of Fourier Transform")
121     plt.xlabel("n")
122     plt.ylabel("F$_n$")
123     plt.legend()
124     plt.savefig("images/exercise3_0.45_components_"+str(h)+".png", dpi
    =300)
125     plt.show()
126
127     ###################
128     # BACK TRANSFORM #
129     ###################
130
131     def realF(n):
132         return fReal(func, N, n, h)
133
134     def imagF(n):
135         return fImag(func, N, n, h)
136
137     back_real = np.array([])
138     back_imag = np.array([])
139
140     for i in m:
141         back_real = np.append(back_real, fmReal(realF, imagF, N, i))
```

```
142         back_imag = np.append(back_imag, fmImag(realF, imagF, N, i))
143 
144     plt.title("The reconstructed function and from the Fourier back-
    transform  \n for h = " + h_labels[x] + " and the original function
    ")
145     plt.xlabel("t")
146     plt.ylabel("f(t)")
147     plt.plot(t_list, func(t_list), label="Original function", color="
    blue")
148     plt.plot(tm, back_real, label="Fourier back-transform (real)",
    color="red")
149     plt.plot(tm, back_imag, label="Fourier back-transform (imaginary)",
     color="yellow")
150     plt.legend()
151     plt.savefig("images/exercise3_0.45_back_"+str(h)+".png", dpi=300)
152     plt.show()
153 
154     plt.title("The difference between the reconstructed and \n original
     functions for h = " + h_labels[x])
155     plt.plot(tm, func(tm) - back_real)
156     plt.xlabel("t")
157     plt.ylabel("error")
158     plt.savefig("images/exercise3_0.45_error_"+str(h)+".png", dpi=300)
159     plt.show()
160 
161     x += 1
162 
163 ##################
164 #                #
165 #   cos(6pit)    #
166 #                #
167 ##################
168 
169 
170 # FUNCTION BEING ANALYSED
171 def func(t):
172     return np.cos(6*np.pi*t)
173 
174 
175 # LOOP OVER h VALUES
176 h_list = np.array([0.6, 0.2, 0.1, 0.04])
177 
178 # INITIAL VARIABLES
179 N = 32
180 m = np.arange(0, N, 1)
181 n = np.arange(1, N+1, 1)
182 
183 for h in h_list:
184     ##############
185     # TRANSFORM #
186     ##############
187 
```

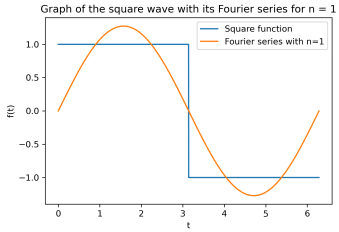
```
188     N = 32 # number of samples
189 
190     # SAMPLING RATE
191     rate = 1/h
192     print("The sampling rate is " + str(rate) + " samples per second")
193 
194     # FUNDAMENTAL FREQUENCY
195     omega1 = 2*np.pi/(h*N)
196     print("The fundamental frequency is " + str(omega1) + " /s")
197 
198     # PLOT THE FUNCTION
199     t_list = np.linspace(0, (N-1) * h, 1000)
200     plt.plot(t_list, func(t_list))
201 
202     # SAMPLE TIMES
203     tm = m*h
204 
205     # ADD SAMPLED POINTS
206     plt.scatter(tm, func(tm), marker='x', color='black')
207     plt.xlabel("t")
208     plt.ylabel("f(t)")
209     plt.title("The graph of the function cos(6$\pi$t) for h = " + str(h
    ))
210     plt.savefig("images/exercise3_0.6_graph_"+str(h)+".png", dpi=300)
211     plt.show()
212 
213     real = np.array([])
214     imag = np.array([])
215 
216     for i in n:
217         real = np.append(real, fReal(func, N, i, h))
218         imag = np.append(imag, fImag(func, N, i, h))
219 
220     plt.title("The components of the Fourier Transform of cos(6$\pi$t)
    for h = " + str(h))
221     plt.plot(n, real, label="Real component of Fourier Transform")
222     plt.plot(n, imag, label="Imaginary component of Fourier Transform")
223     plt.xlabel("n")
224     plt.legend()
225     plt.savefig("images/exercise3_0.6_components_"+str(h)+".png", dpi
    =300)
226     plt.show()
227 
228     ###################
229     # POWER SPECTRUM #
230     ###################
231 
232     def powerSpec(func, N, n, h):
233         return fReal(func, N, n, h)**2 + fImag(func, N, n, h)**2
234 
235     plt.title("Power spectrum for h = " + str(h))
236     plt.plot(n, powerSpec(func, N, n, h))
```

```
    plt.xlabel("n")
    plt.savefig("images/exercise3_0.6_power_"+str(h)+".png", dpi=300)
    plt.show()

    ###################
    # BACK TRANSFORM #
    ###################

    def realF(n):
        return fReal(func, N, n, h)

    def imagF(n):
        return fImag(func, N, n, h)

    back_real = np.array([])
    back_imag = np.array([])

    for i in m:
        back_real = np.append(back_real, fmReal(realF, imagF, N, i))
        back_imag = np.append(back_imag, fmImag(realF, imagF, N, i))

    plt.title("The reconstructed function and from the Fourier back-
    transform  \n for h = " + str(h) + " and the original function")
    plt.xlabel("t")
    plt.ylabel("f(t)")
    plt.plot(t_list, func(t_list), label="Original function", color="
    blue")
    plt.plot(tm, back_real, label="Fourier back-transform (real)",
    color="red")
    plt.plot(tm, back_imag, label="Fourier back-transform (imaginary)",
     color="yellow")
    plt.legend()
    plt.savefig("images/exercise3_0.6_back_"+str(h)+".png", dpi=300)
    plt.show()

    plt.title("The difference between the reconstructed and \n original
     functions for h = " + str(h))
    plt.plot(tm, func(tm) - back_real)
    plt.xlabel("t")
    plt.ylabel("error")
    plt.savefig("images/exercise3_0.6_error_"+str(h)+".png", dpi=300)
    plt.show()
```

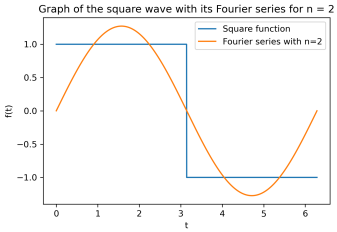
5.3 Appendix C: Extra Graphs

These graphs while useful to look at are not strictly necessary to understand that as the number of Fourier coefficients, n, is increased the Fourier series better approximates the original signal.

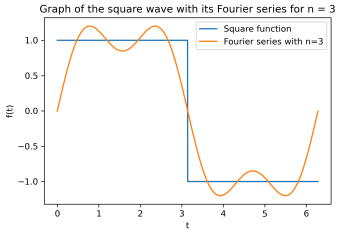
The graphs in figures 5.2 and 5.3 simply demonstrate the behaviour of the square and rectangular waves and their Fourier series approximations over timescales longer than a single period. Which is useful to see, but again not strictly necessary.



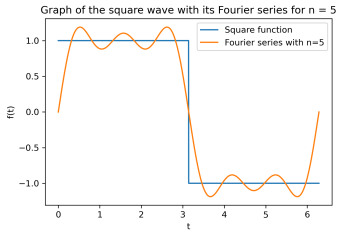
(a) The graph of the square wave function with its Fourier series evaluated for $n = 1$.



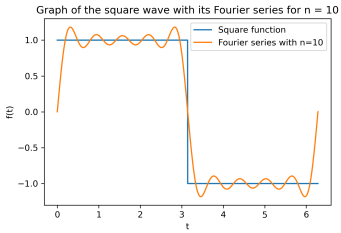
(b) The graph of the square wave function with its Fourier series evaluated for $n = 2$.



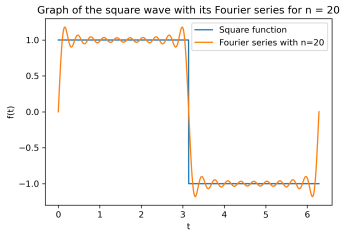
(c) The graph of the square wave function with its Fourier series evaluated for $n = 3$.



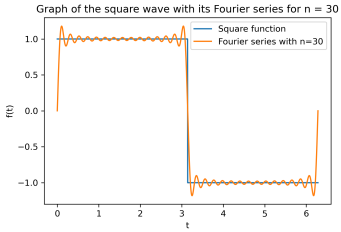
(d) The graph of the square wave function with its Fourier series evaluated for $n = 5$.



(e) The graph of the square wave function with its Fourier series evaluated for $n = 20$.



(f) The graph of the square wave function with its Fourier series evaluated for $n = 20$.



(g) The graph of the square wave function with its Fourier series evaluated for $n = 30$.

Figure 5.1: The graph of the square wave function with its Fourier series evaluated for varying values of n.

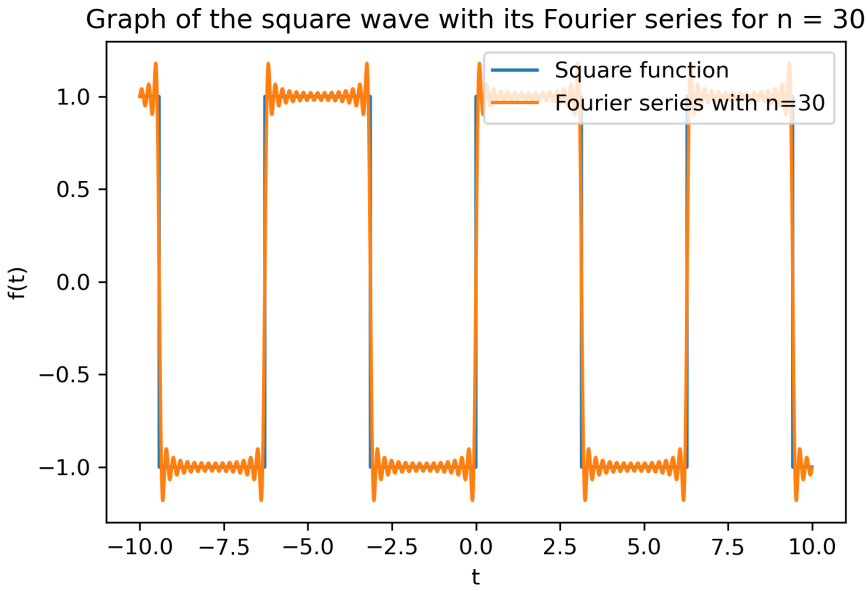


Figure 5.2: The graph of the square wave function with its Fourier series evaluated for $n = 30$ on the interval $[-10, 10]$.

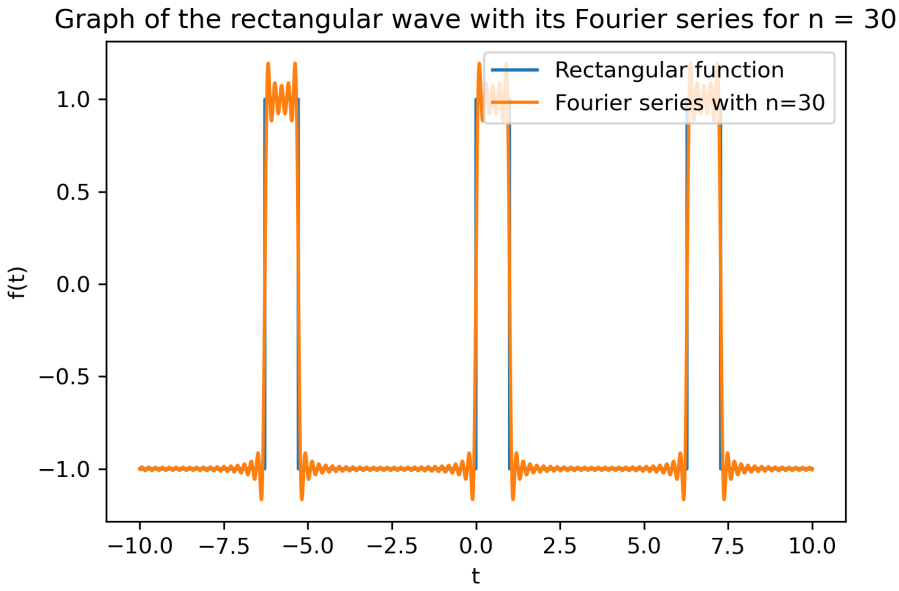
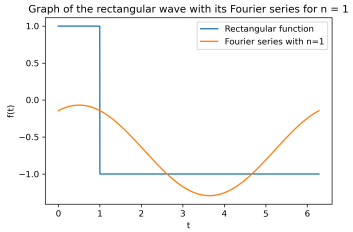
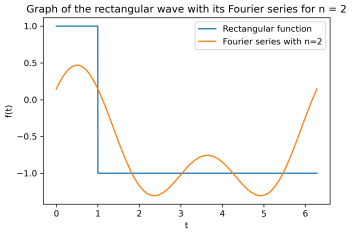


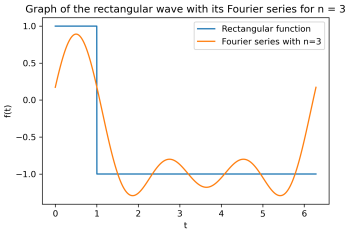
Figure 5.3: The graph of the rectangular wave function with its Fourier series evaluated for $n = 30$ on the interval $[-10, 10]$.



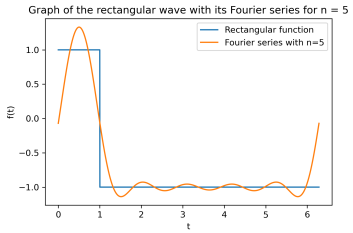
(a) The graph of the rectangular wave function with its Fourier series evaluated for $n = 1$.



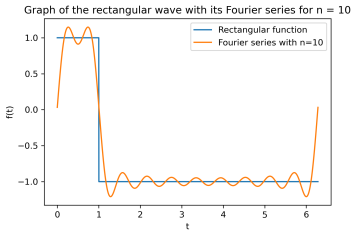
(b) The graph of the rectangular wave function with its Fourier series evaluated for $n = 2$.



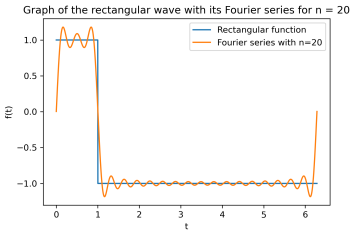
(c) The graph of the rectangular wave function with its Fourier series evaluated for $n = 3$.



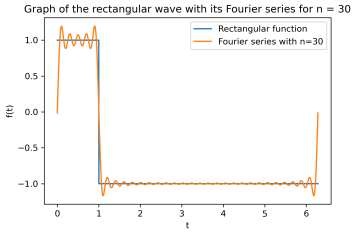
(d) The graph of the rectangular wave function with its Fourier series evaluated for $n = 5$.



(e) The graph of the rectangular wave function with its Fourier series evaluated for $n = 20$.



(f) The graph of the rectangular wave function with its Fourier series evaluated for $n = 20$.



(g) The graph of the rectangular wave function with its Fourier series evaluated for $n = 30$.

Figure 5.4: The graph of the rectangular wave function with its Fourier series evaluated for varying values of n.