JOHNSON MATTHEY
TECHNOLOGY REVIEW

www.technology.matthey.com

Basics of Fourier Analysis of Time Series Data

A practical guide to use of the Fourier transform in an industrial setting

Carl Tipton
Johnson Matthey, PO Box 1, Chilton Office, Belasis Avenue, Billingham, TS23 1LB, UK

Email: carl.tipton@matthey.com

NON-PEER REVIEWED FEATURE

Received 18th February 2021; Online 8th March 2022

1. Introduction

There are few mathematical breakthroughs that have had as dramatic impact on the scientific process as the Fourier transform. Defined in 1807 in a paper by Jean Baptiste Joseph Fourier (1) to solve a problem in heat conduction, the integral transform, Equation (i):

$$G(\omega) = \int_{-\infty}^{\infty} e^{i\omega t} g(t)dt \qquad \text{(i)}$$

and its inverse, Equation (ii):

$$g(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-i\omega t} G(\omega)d\omega \qquad \text{(ii)}$$

provide the framework to determine the spectral make up of a time varying function *g(t)* using Equation (i). Conversely, if the frequency domain is understood $G(\omega)$, the time signal can be derived using Equation (ii). The same analysis can be applied to spatial functions to yield wave number spectra and is the basis for a significant portion of wave optics, and is used in techniques such as Fourier transform infrared (FTIR) spectroscopy (2).

The transform, which is part of a wider family of integral transforms (3), had a profound impact on the development of much of 19th and 20th century mathematical physics. Previously intractable problems in optics, electromagnetism and acoustics became soluble. The insights these breakthroughs yielded paved the way for quantum mechanics and much of modern science. The famous Heisenberg uncertainty principle is actually just a mathematical property of the Fourier transform in Schrödinger's wave mechanics (4). Domínguez gives a good overview of this history and some of the mathematical properties of the transform that make it so useful (5).

A significant hurdle with the practical application of the Fourier transform in real-world problems is that it is mathematically challenging to calculate for even the simplest of functions. As a consequence the transform is not taught in the UK until undergraduate level and even then only in mathematically heavy courses such as mathematics, physics and engineering. To make progress in practical problems numerical methods are generally required, meaning the practical application of the Fourier transform can feel like an esoteric part of computer science, rather than the scientific core of the modern world.

Fortunately, the great leaps in understanding that quantum mechanics gave us in electronics has ultimately led to a situation where anyone who wants to, can with a few lines of Python (6) code use sophisticated algorithms that have been developed in the post-World War II period. As such, calculations of the Fourier transform are readily available to those that would like to make use of them.

Unfortunately, the education around how to do practical Fourier analysis has become something of a dark art, which is often picked up in an *ad hoc* manner in postgraduate studies. The advent of accessible artificial intelligence algorithms has

© 2022 Johnson Matthey

further obscured the basic techniques of Fourier analysis and created a strange scenario where even basic spectral methods are being conducted with inefficient computationally heavy neural network approaches.

In this short article we outline some basic practical steps for successfully conducting Fourier analysis. We will also give a few example Python scripts so the interested reader may apply these techniques to their data.

2. The Discrete Fourier Transform

The first challenge for any numerical method is the digitisation step during which the smooth curves of analytical functions must be turned into discrete numbers. There are two sources of data that are normally digitised:

- Analytic functions
- Experimental time series.

Discussing these in turn, when an analytic function of time $g(t)$ is evaluated, it is relatively trivial to generate the discretised function with N samples in the time window $0 < t \leq T$ (Equations (iii) and (iv)):

$$g_j = g(j\delta) \quad \{j \in \mathbb{N}, 0 < j \leq N\} \tag{iii}$$

where

$$\delta = \frac{T}{N} \tag{iv}$$

The numerical value of δ is of crucial importance in numerical estimates of the Fourier transform. It places limits on what information is lost in the discretisation and plays a fundamental role in how experimental work should be designed. It is more usual to quote its reciprocal, which is the sampling frequency, f_s (Equation (v)):

$$f_S = \frac{1}{\delta} \tag{v}$$

It is this frequency that appears in one of the most important results associated with the Fourier transform: Nyquist's theorem (7). This result states (Equation (vi)):

$$f_S > 2B \tag{vi}$$

where B is the highest frequency component in the signal in $g(t)$.

Nyquist's theorem is particularly important as we turn our discussion to sampling experimental data. In theoretical work one can choose, in principle, δ to be as small as is necessary. However, in experimental work this is not an available option; the cost of data loggers increases significantly with the sampling frequency and data storage problems quickly become limiting. Moreover, in nearly all applications where data is recorded by a computer, signals are voltages recorded by an analogue to digital converter (ADC). To conduct scientific work a 12 bit ADC is the standard level. This means that a voltage signal varying between a nominal full-scale deflection ±10 V is recorded to the nearest 5 mV as defined in Equation (vii):

$$\delta V = 2 \times 10 \times 2^{-12} \approx 5 \times 10^{-3}\,\text{V} \tag{vii}$$

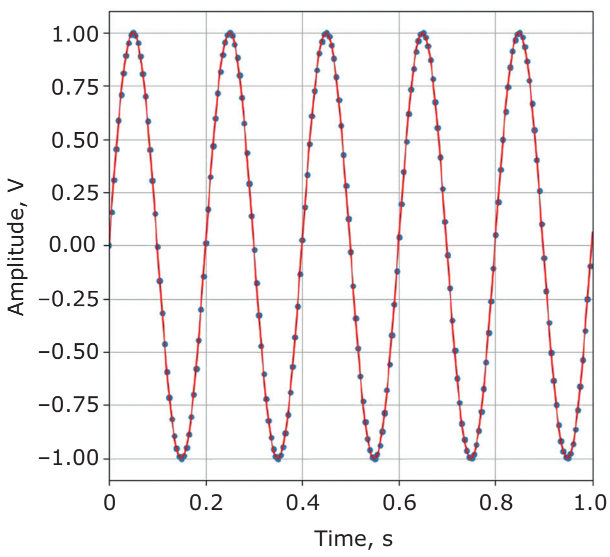
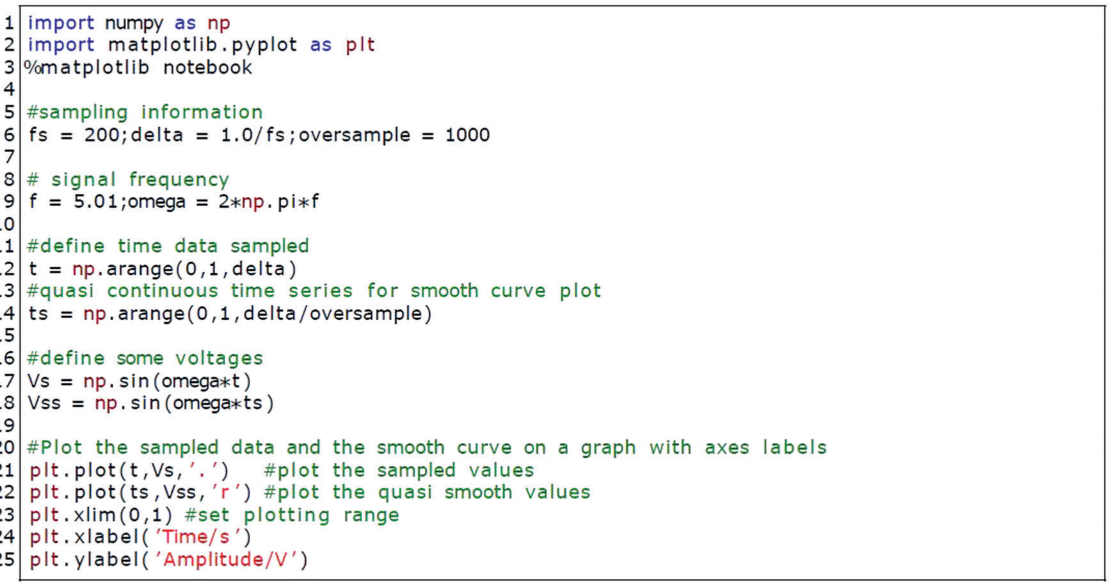


Fig. 1. Example of a sampled sine curve. The dots denote the sampled data, the red curve the analytic values

When numerical results are compared to experimental results this level of precision must always be borne in mind, as the limitations of the sampling frequency or the voltage level are both likely to be significantly more coarse grain in the experimental work. An example of the effects of this digitisation step is shown in **Figure 1**. A 5.01 Hz sine wave has been sampled for 1 s, with a sampling frequency of 200 Hz. The blue dots denote the locations of the sampled data and the red curve the analytic form of a sine curve with this frequency.

The popular data analytics tool Jupyter (8) was used to generate the graph shown in **Figure 1**, this is part of the open-source data analytics bundle Anaconda. The code used is shown in **Figure 2**. The majority of the code is presentational and associated with plotting the graph using the Python library matplotlib (9). However, the numerical analysis makes use of the versatile NumPy library (10). The key lines for our discussion are lines 17



```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3 %matplotlib notebook
 4
 5 #sampling information
 6 fs = 200;delta = 1.0/fs;oversample = 1000
 7
 8 # signal frequency
 9 f = 5.01;omega = 2*np.pi*f
10
11 #define time data sampled
12 t = np.arange(0,1,delta)
13 #quasi continuous time series for smooth curve plot
14 ts = np.arange(0,1,delta/oversample)
15
16 #define some voltages
17 Vs = np.sin(omega*t)
18 Vss = np.sin(omega*ts)
19
20 #Plot the sampled data and the smooth curve on a graph with axes labels
21 plt.plot(t,Vs,'.')   #plot the sampled values
22 plt.plot(ts,Vss,'r') #plot the quasi smooth values
23 plt.xlim(0,1) #set plotting range
24 plt.xlabel('Time/s')
25 plt.ylabel('Amplitude/V')
```

Fig. 2. Python code used to generate **Figure 1**

and 18, which generate two vectors Vs and Vss. The vector Vss is the smooth underlying 5.01 Hz sinusoidal signal and Vs is the signal sampled with a sampling frequency of 200 Hz. It is these two vectors that are manipulated in the sections that follow.

3. The Fast Fourier Transform

Having defined the digitised signal, discrete Fourier transform (DFT) can be defined as shown in Equations (viii) and (ix):

$$G_k = \sum_{j}^{N-1} g_j e^{i\theta jk} \qquad \text{(viii)}$$

where

$$\theta = \frac{2\pi}{N} \qquad \text{(ix)}$$

The DFT is simple enough to code from first principles that it is often used as an example numerical problem to teach students how to use loops in a given programming language, however it is rarely used in production code because it is computationally inefficient. As the number of samples increases, the number of calculations increases with the square of the number of samples ($O(N^2)$). If this efficiency problem had not been solved in a paper by Cooley and Tukey (11), where they introduced what is known as the fast Fourier transform (FFT), a significant amount of the telecommunications sector would not have been possible. The algorithm they published was actually first discovered by Gauss in 1809 in an unpublished paper and uses a divide and conquer technique. The original time series is split into odd samples and even samples; and then a recursive approach used to construct the Fourier spectrum. This is the reason that many implementations of this algorithm impose the restriction that the number of samples should be a power of two, as this improves the operational efficiency. The efficiency of the FFT scales as $O(N \log N)$ opened up the possibility of using Fourier analysis in technical areas that previously would not have been possible.

It is not an exaggeration to say the FFT revolutionised electronic engineering and in turn computer science. Nearly all digital communications rely on the FFT in some form. A measure of how integral to the mathematical sciences the algorithm has become is that improvements to the algorithm continue to the modern day, for example a particularly fast and robust implementation of the FFT called the 'fastest Fourier transform in the West' (FFTW) was developed and maintained by academics at the Massachusetts Institute of Technology (MIT), USA (12), and remains an active project. Despite how readily available FFT algorithms have become it is still easy to make mistakes when using them in a real-world example. A raw power spectrum of the time series shown in **Figure 1** is shown in **Figure 3**. The spectrum is shown on a log scale to highlight the detailed features that might otherwise be missed.

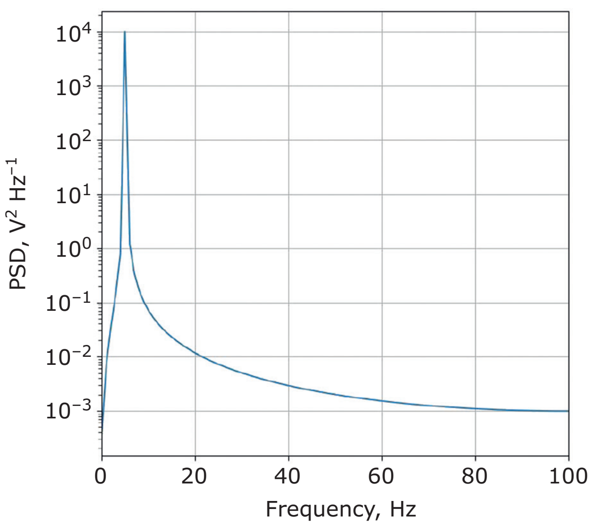


Fig. 3. The raw power spectrum of the sampled time series in **Figure 1**

The first and most important point is that the spectrum plotted is actually a power spectrum. Theoretically this is defined as Equation (x):

$$P_k = G_k^* G_k = \|G_k\|^2 \qquad (x)$$

where the G_k^* is the complex conjugate of each Fourier component. The process of finding the power spectrum is lossy, as all phase information in the signal is lost. Despite this, there are many situations where the power spectrum is a much more useful quantity than the raw time series. In this example the large peak at 5.01 Hz, which is seven orders of magnitude above the noise floor, easily identifies the main frequency present in original times series. The code snippet in **Figure 4** illustrates how simple using the FFT is with a modern analytics package like Jupyter. Line 2 takes the sampled data Vs from **Figure 2**, calculates the FFT and converts it into a power spectrum (by taking the absolute value and squaring each component of the vector). Line 3 is simply the calculation of the frequency associated with each bin in the spectrum and is determined by the original sampling frequency fs of the signal. The remainder of the snippet is about presenting the spectrum on a graph.

4. Implementation of Fast Fourier Transform

The ideal nature of the original time series used to calculate the power spectrum shown in **Figure 3** obfuscates some of the limitations of this naïve brute force use of the FFT. A typical experimental time series has underlying electrical noise and the time digitisation further distorts the signal. In the following sections we shall discuss the best practice that should be followed to get the best estimate of a power spectrum from an experimental signal. We first simulate what a noisy experimental signal might look like by adding Gaussian noise and then splitting the data into 20 different finite levels to simulate the effect of an analogue to digital converter. The three signals are shown in **Figure 5**. The digitised noisy signal is representative of many experimental signals met in practice.

The main challenge with any experimental setup is designing the experiment to give the best answers we can reasonably expect. The processing of a time series to give the most spectral insight is no different. In this section we will attempt to give some basic guidelines that a novice time series analyst should follow, where possible, when conducting spectral analysis.

4.1 Filter High Frequency Signals

The time series we are analysing nominally has a single harmonic component at 5.01 Hz. Nyquist's theorem guides us as to what sampling frequency should be used. The 200 Hz sampling frequency used in **Figures 1** and **3** is too high to get good details in the frequency range of interest. If we assume that we are interested in only whether the first two harmonics are present, then the sampling frequency should at most only be 40 Hz. This figure was arrived at by assuming the

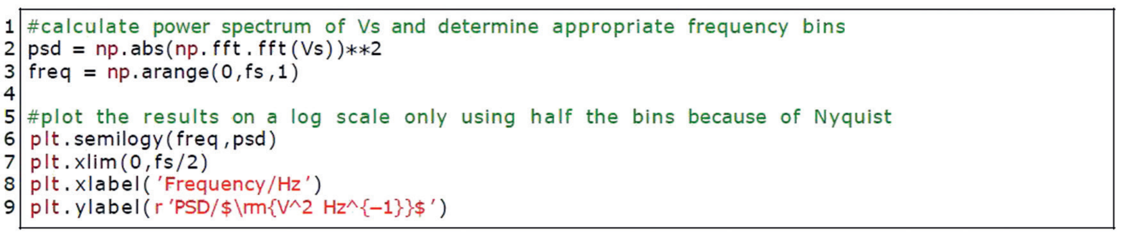

```
1 #calculate power spectrum of Vs and determine appropriate frequency bins
2 psd = np.abs(np.fft.fft(Vs))**2
3 freq = np.arange(0,fs,1)
4
5 #plot the results on a log scale only using half the bins because of Nyquist
6 plt.semilogy(freq,psd)
7 plt.xlim(0,fs/2)
8 plt.xlabel('Frequency/Hz')
9 plt.ylabel(r'PSD/$\rm{V^2 Hz^{-1}}$')
```

Fig. 4. Python code used to generate **Figure 3**

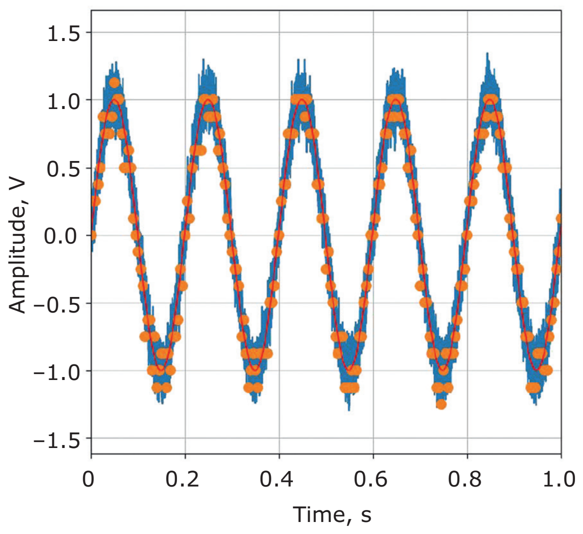


Fig. 5. The red curve is a theoretical sine wave. Gaussian noise has been added to this signal (blue signal) and finally this noisy signal has been digitised to simulate the effect of a coarse analogue to digital converter (orange dots). A sampling frequency of 200 Hz has been used

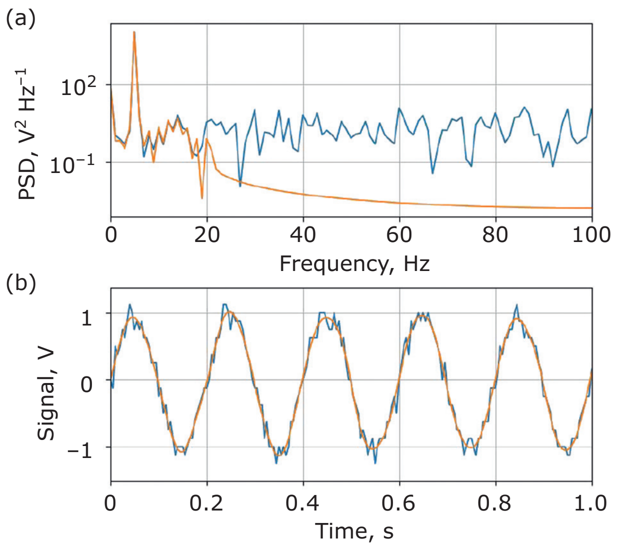


Fig. 6. (a) The raw power spectrum of the noisy sampled time series shown in **Figure 5** before (blue trace) and after a low pass filter (orange trace) is applied to the signal; (b) the impact of applying the low pass filter

fundamental is at 5 Hz then the third harmonic is at 20 Hz. Nyquist implies we should then double this value. However, another consequence of the Nyquist theorem is that if a signal contains frequency components that are above the Nyquist frequency, for example due to electronic noise, then the FFT algorithm breaks down and higher frequencies are erroneously folded back into the low frequency bins.

Most ADC systems have some form of low pass filter that stops very high frequency noise being recorded. However, these filters are unlikely to be set at the correct frequency for any given application. An option that can be used if the raw data has been sampled at a sufficiently high frequency is to apply a low pass digital filter with a critical frequency sufficiently above the range of interest. It is common to choose a filter at the desired Nyquist frequency, in our case 20 Hz. The impact of applying such a filter is illustrated in **Figure 6**. Prior to applying the low pass filter there are components at higher frequencies that have the potential to obscure the underlying data. A similar effect can be achieved by using a separate electronic low pass filter to the experimental set up, again with the critical filter frequency set at the Nyquist frequency.

The code to apply the filter used here is shown in **Figure 7**. The vector Vs2 is the noisy sampled time series data shown in **Figure 5** and the returned filtered signal Vs3 is the smoothed signal. We have used a simple Butterworth filter (13) as an example but there are many others available in the Python toolbox SciPy (14).

4.2 Downsampling

Once a low pass filter has been applied to the signal it is sensible to resample the data at the lower frequency to enable more details of the spectrum to be resolved in the region of interest. This process is called downsampling (15) and should only be done if there are no higher frequency components that are likely to interfere with the results. Since we have applied a low pass filter there are no higher components in the time series data, hence downsampling can be applied safely. The reasons for doing this are perhaps not obvious at first sight, but as discussed in the next section, the computational impact of having an oversampled time series can be significant, particularly when fine frequency resolution is required in the power spectrum.

4.3 Extend the Sampling Window

If one considers two notes of frequency f_1 and f_2 which are played at the same time, a third lower frequency can be heard. This is called a beat frequency, f_b (Equation (xi)):

$$f_b = f_1 - f_2 = \Delta f \qquad \text{(xi)}$$

```python
from scipy import signal

Wn =0.2
b, a = signal.butter(20, Wn, 'low')
Vs3 = signal.filtfilt(b, a, Vs2)
```

Fig. 7. Python code used to filter the noisy data shown in **Figure 5**

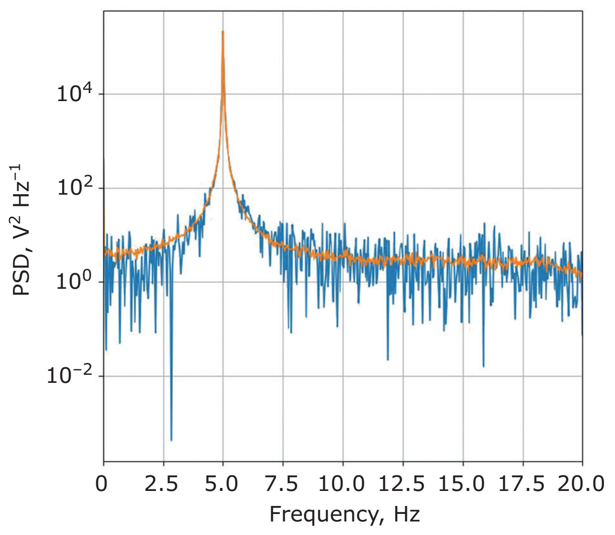


Fig. 8. The raw power spectrum of the filtered time series shown in **Figure 6** with a reduced sampling frequency for a single extended time window 25.8 s (blue spectra). The effect of averaging multiple spectra using Welch's method is shown in the orange spectra

If the notes are nearly the same frequency, the beat frequency becomes very small, vanishing to zero when they are identical. Guitarists sometimes use this effect to tune their instruments. This point illustrates that in order to distinguish between two frequencies of slightly different tones, the frequency resolution is limited by the length of the time series recorded. To increase frequency resolution one must record longer time series. The impact of increasing the sampling time frame can be quite dramatic. The power spectrum shown in **Figure 8** is what is obtained if 25.8 s of data are used at the 40 Hz sampling frequency (1024 data points). The fundamental peak at 5.01 Hz is much sharper allowing for a better resolution of the frequency.

If the downsampling step had not been performed to get the same resolution, the number of data points included in the FFT would need to be increased five-fold for no benefit. It is tempting to simply take very long time series and then calculate the power spectrum with a very large number of data points. However, this can be counter productive, not to say computationally inefficient. The spectrum shown in **Figure 8** has 512 different frequency bins for $0 < f < 0.5f_s$, which gives a resolution of Equation (xii):

$$\Delta f = \frac{20}{512} \approx 0.04\text{Hz} \qquad \text{(xii)}$$

If a frequency resolution finer than this is required then it is reasonable to use longer time series. However, fine resolution bins can lead to difficult-to-interpret noise floors. It is unlikely, for example, that there is a two order of magnitude difference in the power content of two adjacent bins outside of the main harmonics of any time series, yet that is what the blue spectrum shown in **Figure 8** indicates. This wildly oscillating noise floor is an artefact of the discretisation, rather than a true reflection of the noise content of the signal.

4.4 Averaging Spectra and Window Functions

If one has the luxury of very long time series data being available, it is good practice to calculate multiple power spectra by splitting the data into separate time windows, and then reporting the mean result for each frequency bin. This is akin to conducting an experimental measurement multiple times and then reporting the mean results. This was first introduced by Bartlett (16, 17) and improved on by Welch (18) who introduced the idea of overlapping windows to reduce edge effects of the windows. The impact of averaging is illustrated by the orange power spectrum shown in **Figure 8**. This spectrum is the average of 39 separate spectra. The noise reduction is significant and variation between adjacent bins is significantly smaller.

The final improvement to our experimental power spectrum we will discuss is to use a nonrectangular window function. The mathematical underpinning of the FFT assumes that the time series repeats for all time. As such, the finite time length has consequences on the shape of the power spectrum. The power spectrum of a box car window is convolved with the power spectrum of the repeating

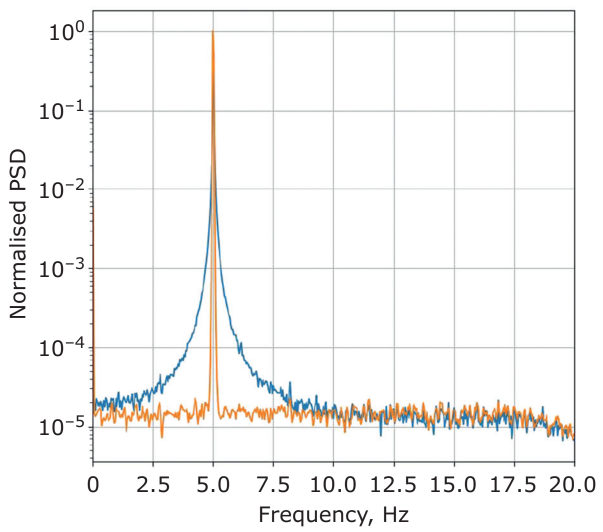


Fig. 9. Normalised power spectra using a box car window (blue) and a Hann window (orange). The peak is much sharper using the Hann window function so is better for discriminating nearby frequencies

time series. Depending on the application a box car window is unlikely to be the best window to use. There are many windows available that may be more appropriate. Here we use the Hann (15) window to illustrate the point. The normalised power spectrum with a box car window and the Hann window is shown in **Figure 9**. The peak near 5 Hz is much narrower with the window function applied. This means that a better frequency resolution is achieved. The cost for this is that the amplitude information in the signal is distorted; the two signals have been normalised to the peak to assist in the comparison.

A function to bring together; the low pass filtering, the downsampling, the averaging and the incorporation of a Hann window is shown in **Figure 10**. This short function illustrates how easily all the ideas can be brought together using a modern data analytics language such as Python.

5. Conclusions

The application of the FFT to data is one of the most widespread numerical algorithms. It is integral to a huge amount of fundamental scientific research and engineering. In an industrial setting the power spectrum is used as a noise reduction method on many sensors, in the communication sector information is compressed using the FFT and in the laboratory many measurement techniques intrinsically make use of the FFT.

Many instruments report spectra directly, for example the output of an FTIR spectrometer, but it is always prudent to understand what analysis is being conducted on our behalf. As outlined here many analytical steps are happening and they may not be applicable to the analysis that we wish to conduct. Fortunately, many numerical packages are readily available that we as users can use to undertake our own Fourier analysis. All the graphs presented in this article have been generated from within a Jupyter notebook using the standard Python libraries bundled with Anaconda. These

```python
import numpy as np
from scipy import signal

#Funtion to per averaged PSD with Hann window
def NormFiltDownSample(Vs2):
  # filter Vs2 the experimental data
  Wn =0.2
  b, a = signal.butter(20, Wn, 'low')
  Vs3 = signal.filtfilt(b, a, Vs2)

  # downsample by taking every 5th sample
  Vs4 = Vs3[::5]

  #intialise average psd vector to 0
  psdw = Vs4[0:1024]*0

  window = np.hanning(1024)
  # loop through 39 windows to get average PSD
  loops = 39
  for i in range(loops):
    psdw = abs(np.fft.fft(0.3*Vs4[i*512:i*512+1024]*window))**2 + psdw

  return psdw/max(psdw)
```

Fig. 10. Python code bringing together low pass filtering, downsampling, a Hann window function and spectral averaging

are readily available tools that we can all use if we have the inclination. Moreover, any time series can be analysed using Fourier analysis to reveal any possible underlying periodic behaviour. Atypical examples might be timesheets, holidays and production data.

The first stage of data analysis for nearly all time series data should be to understand the power spectra. The first step for a novice is to download the Anaconda bundle and start up the Jupyter executable, the second step is to search one of the many online tutorials (for example, (19)) in data analysis in Python and start experimenting. We are fortunate to live in an age when data analysis is an exceptionally easy thing to do. Let us all embrace this gift!

References

1. J. B. J. Fourier, 'Théorie de la Propagation de la Chaleur dans les Solides', 21st December, 1807, *Manuscript submitted to the Institute of France*
2. P. R. Griffiths and J. A de Haseth, "Fourier Transform Infrared Spectrometry", 2nd Edn., John Wiley & Sons Inc, Hoboken, USA, 2007, 560 pp
3. K. P. Das, "Integral Transforms and their Applications", Alpha Science International Ltd, Oxford, UK, 2019, 224 pp
4. A. I. M. Rae and J. Napolitano, "Quantum Mechanics", 6th Edn., Taylor and Francis Group LLC, Boca Raton, USA, 2016, 440 pp
5. A. Domínguez, *IEEE Pulse*, 2016, **7**, (1), 53
6. G. van Rossum and F. L. Drake, "Python 3: Reference Manual", Part 2, CreateSpace, Scotts Valley, USA, 2009
7. H. Nyquist, *Trans. Am. Inst. Electr. Eng.*, 1928, **47**, (2), 617
8. T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing and Jupyter Development Team, 'Jupyter Notebooks – A Publishing Format for Reproducible Computational Workflows', in "Positioning and Power in Academic Publishing: Players, Agents and Agendas", eds. F. Loizides and B. Schmidt, IOS Press, Amsterdam, The Netherlands, 2016, pp. 87–90
9. J. D. Hunter, *Comput. Sci. Eng.*, 2007, **9**, (3), 90
10. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, *Nature*, 2020, **585**, (7825), 357
11. J. W. Cooley and J. W. Tukey, *Math. Comp.*, 1965, **19**, (90), 297
12. M. Frigo and S. G. Johnson, *Proc. IEEE*, 2005, **93**, (2), 216
13. S. Butterworth, *Exper. Wire. Wire. Eng.*, 1930, **7**, 536
14. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, *Nat. Methods*, 2020, **17**, (3), 352
15. Alan V. Oppenheim, Ronald W. Schafer and John R. Buck, "DiscreteTime Signal Pro cessing*",* 2nd Edn., Prentice-Hall Inc, Upper Saddle River, USA, 1999
16. M. S. Bartlett, *Nature*, 1948, **161**, (4096), 686
17. M. S. Bartlett, *Biometrika*, 1950, **37**, (1–2), 1
18. P. Welch, *IEEE Trans. Audio Electroacoust.*, 1967, **15**, (2), 70
19. B. Pryke, 'How to Use Jupyter Notebook in 2020: A Beginner's Tutorial', Dataquest Labs Inc, Sommerville, USA, 24th August, 2020

The Author

Carl Tipton gained his PhD in Nonlinear Physics from the University of Manchester, UK, in 2003. Since then, he worked as a Development Physicist at Tracerco, UK. He is currently a Measurement Engineer at Johnson Matthey, Chilton, UK, where he works to optimise Johnson Matthey industrial processes.