

Fourier Transform and its Applications

Teo Feliu
April 29, 2022

Data is everywhere. The ease of storing information when it is stored in numbers is what makes data so widespread. However, manipulating, comparing, and transporting large sets of bits is not particularly easy or possible. Things like processing images and streaming large sets of data is close to impossible when there are billions of bits, which is why we rely on transforming this data to make it more comprehensible and manageable. Fourier series or transform is a technique used to retain information in a function or set of data points, and simply maps the data from the time domain to the frequency domain. At first glance, this might not be seen as particularly useful, but Fourier transform makes a multitude of daily essential procedures fast and possible. A Fourier series maps a data into a periodic sinusoid function on the frequency domain, and transform expands this onto the infinite domain. Storing and displaying data in a frequency magnitude spectrum improves data manipulation and interpretation like compression and processing, and has several applications within Mathematics, Physics, Computer Science, and beyond that make essential procedures possible.

Joseph Fourier was a French orphan with aspirations of making the French artillery, but was rejected and sent to Seminary school to be a priest. He was arrested for supporting the French Revolution, but the revolution ended before he was set to be executed, so not only was his life spared, but he was honored by a chair at the Ecole Polytechnique in Paris, one of France's most selective schools, where he succeeded Joseph-Louis Lagrange. Fourier was

arrested a few times more, once again narrowly avoiding the Guillotine on some instances. Joseph Fourier caught Napoleon's eye, which eventually led him to his election to the Academie des Sciences. This election, however, was vetoed by Louis XVII. Fourier constructed a lot of research on the theory of heat, including Fourier systems and the sine series, which Lagrange, his doctoral advisor, rejected. He also discovered the greenhouse effect, and modeled this by trapping heat by wrapping himself in blankets. Fourier's bad luck followed him to his grave, where his "greenhouse effect" made him trip over his blankets and fall down his home's stairs. Joseph Fourier died not knowing about his invention of Fourier Transform, since he developed Fourier Transform to help him solve the heat equation, but is now used for all kinds of functions.

Fourier systems provide the ability to represent functions as linear combinations of sinusoids, or functions made up of sines and cosines. These functions are represented over a period, say of size N . A key characteristic of a Fourier series is that, across the infinite domain, the represented function is a periodic piecewise continuous function, where the interpolated function over period N is repeated over and over. This Fourier Series function has the following equation:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]$$

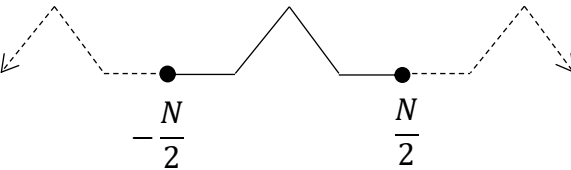
Where scalars a_n and b_n are

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx, \quad n = 0, 1, 2, \dots$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx, \quad n = 0, 1, 2, \dots$$


We've seen a Fourier series adapt a sum of sines and cosines to a function $x(n)$ over a range of N . However, this leaves us with a periodic function that repeats itself every N . Fourier Transform puts a Fourier series in an infinite domain, that is, it transforms periodic function $x(n)$ over an infinite domain.

Series



$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]$$

Transform



Complex sinusoid w.
frequency $\frac{k}{n}$ cycles/sample

$$x(n) = \sum_{k=-\infty}^{\infty} \underbrace{\frac{1}{N} X_k}_{\text{Weight at } \frac{k}{n}} \underbrace{e^{\frac{ik\pi x}{N}}}_{\text{Complex sinusoid w. frequency } \frac{k}{n} \text{ cycles/sample}}$$

Note that in the formula for $x(n)$, we still utilize period length N , as it includes relevant information about our series, even when mapped onto an infinite domain. Essentially, a Fourier Transform takes Fourier series $x(n)$ with period length N and expands N to infinity. Expanding $e^{\frac{ik\pi x}{N}}$ would leave us with a sum of sines and cosines, which we are able to simplify into the shorter term.

The inverse of this equation is used alongside it, where the inputs can be discrete time signals and output frequencies or vice versa. For example, both are used for compression and decompression of images. We will flip this equation to put it in terms of $x[n]$, our discrete time signal, and will find $X[k]$, the frequency spectrum.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-2i\pi \frac{k}{N} n}$$

This will be in complex numbers, but can be plotted by taking the magnitude $|X[k]|$ for every k , called the magnitude spectrum. $X[k]$ will be found and mapped by using Fourier Transform. The first algorithm we will use is one that simply uses the summation seen above and iterates through k and x . Because we are only interested in the portion of the function along period N , we will have $k = (1, 2, \dots, N)$ and $n = (1, 2, \dots, N)$. This rudimentary version of Fourier Transform is called Discrete Fourier Transform (DFT), and because of the double iteration through N , it can be observed that the runtime of DFT is $O(N^2)$.

```
%x=@(n) 2*cos(2*pi*(11/40)*n)  input as a function
x = [0.5 0.5 0 0 0 0 0 0 0]      % input as a set of points
%x=rand(1,100)*10               % input as a set of 100 random points
N=length(x)
X = zeros(1,N)                  % X[g] is the result of the sum for x=g
k=-floor((N-1)/2)              % k is used to center the graph
figure;
hold on;
for g=0:N-1
    for j=0:N-1
        X(g+1)=X(g+1)+x(j+1)*exp(-2i*pi*(g/N)*j)
    end
```

Alg. 1 (DFT)

$X[n] = [(0, 1), (1, 0.904 - 0.293i), (2, 0.654 - 0.475i), (3, 0.345 - 0.475i), (4, 0.095 - 0.293i), (5, 0), (6, 0.095 + 0.293i), (7, 0.345 + 0.475i), (8, 0.654 + 0.475i), (9, 0.904 + 0.293i)]$

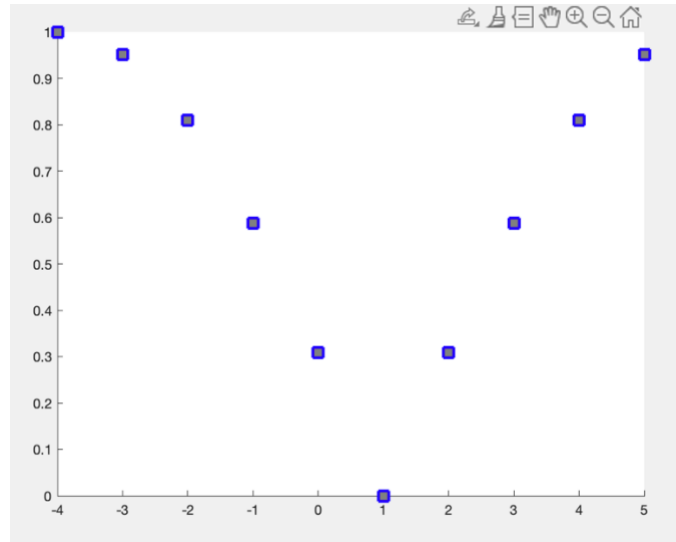


Figure. 1

Because there are only two nonzero values in $x[n]$, this is an example that can easily be verified by hand. Even with such a simple example, it can be seen how tedious this process is, and how quickly the runtime increases as N gets large. By solving for $X[k]$ step by step, it can also be seen that, in the above example, $X[k] = e^{-i\pi\frac{k}{10}} \cos(\frac{\pi k}{10})$. The presence of cosine is apparent in the magnitude spectrum, as seen by the roots and the increase/decrease proportional to a cosine wave.

Let's use a more complex example, where the discrete time signal is a function $x[n] = 2\cos(2\pi\frac{1}{4}n)$ and $N=30$. The fact that $x[n]$ is a continuous function does not change from it being a set of values since $x[n]$ will be indexed at $n=(1, 2, \dots, N)$. The following magnitude spectrum is returned.

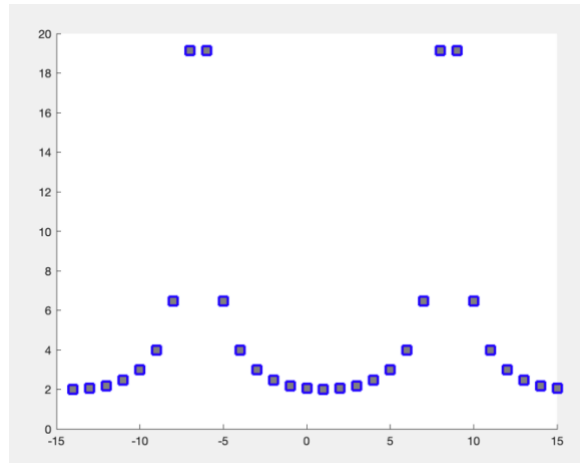


Figure. 2

Here, the frequency of the discrete-time sinusoid $x[n]$ is $1/4$, which is represented using sinusoids with frequencies k/N , so integer multiples of $1/30$ cycles per second. Because $1/4$ is not a multiple of $1/30$, representing $x[n]$ in terms of a complex sinusoid whose frequencies are multiples of $1/30$ will not have a uniform magnitude spectrum. $1/4$ is in between $7/30$ and $8/30$, hence the local maxima around $7/30$ and $8/30$ in the spectrum. The sinusoid that is being represented is closest to those frequencies.

For this last example, the frequencies of $x[n]$ and the representing sinusoid will be more closely related. Let the frequency of $x[n]$ be $9/30$, a multiple of $1/30$. Here, $x[n] = 2\cos(2\pi \frac{9}{30} n)$. We observe the following magnitude spectrum:

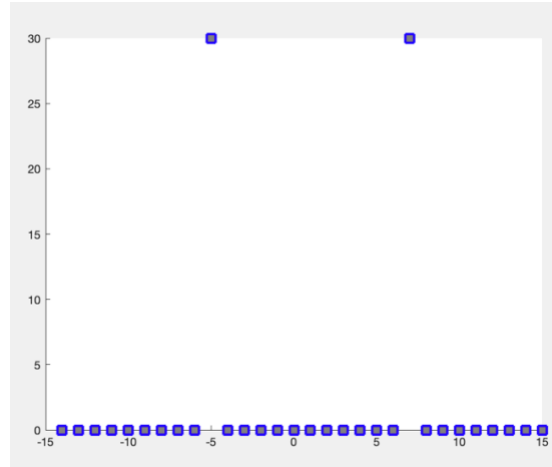


Figure. 3

It can be observed that the acute maxima is at a distance of 9 from the end of N , since $1/30$ is a multiple of $9/30$ by a factor of 9. This shows the effect the correlation between frequencies of the discrete time signal $x[n]$ and that of the representing sinusoid has on the magnitude spectrum.

Using this method to calculate the discrete Fourier Transform is simple and can easily be done by hand, however, DFT is usually performed with extremely large values, where the algorithm would take extremely long. Because of this, a variation of DFT, called Fast Fourier Transform (FFT) is used. FFT calculates DFT in the same way, the difference is in the efficiency of the algorithm. FFT uses an algorithm solving technique called divide and conquer. Divide and conquer is commonly used in sorting algorithms, that make sorting a set of size N with runtime $O(N \log N)$ possible. Similarly, divide and conquer reduces the Fourier Transform runtime to $O(N \log N)$. Divide and conquer works by recursively splitting a larger problem into subproblems, which divide into sub-subproblems, and this goes all the way down until a base case is reached. These subproblems are then merged until the problem is solved.

FFT is an algorithm for computing the discrete Fourier Transform, using a divide and conquer method. Below we see FFT in action, where our $x[n]$ is a set of 8 random values:

```
x = rand(1,8)*10;  
  
N=length(x);  
p=ceil(log2(N));  
N = 2^p;  
N2=N/2;  
WW = exp(-pi*i/N2);  
W=WW.^(0 : N2-1);  
for j=1:p-1  
    t=x(:,1:N2)+x(:,N2+1:N);  
    x=[t ; W.*(x(:,1:N2)-x(:,N2+1:N))];  
    W=[W(:,1:2:N2); W(:,1:2:N2)];  
    N=N2;
```

Alg. 2 (FFT)

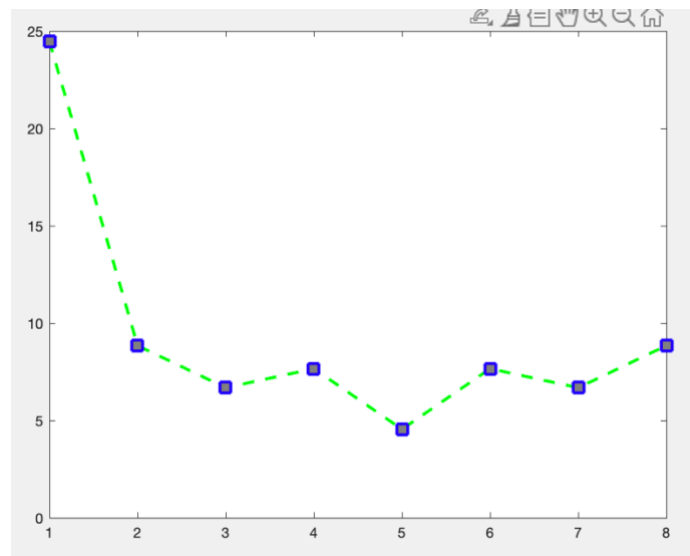


Figure. 4

Usually, in numerical computation, increased error is compromised in order to achieve decreased running time. However, FFT uses the same method of Fourier, the faster runtime is purely due to a more efficient algorithm.

Fourier Transform makes everyday actions fast and possible. One example is data compression. Take JPEG compression. A JPEG image has to be compressed into a smaller amount of data in order to be moved around, as the amount of information would be too large without compression. Say, for example, we want to remove 50% of an image's bits. Removing half of the JPEG's image would obviously not be an effective form of compression, as too much information would be lost and could not be uncompressed back into the original image. Transforming the data values and organizing them in terms of frequency can solve this issue. When dealing with frequencies in a JPEG, the higher frequency components are less important and the lower frequencies are more important. Removing 50% of the higher frequency information could remove, say, only 5% of the encoded image, which is much better for decompression and can preserve the entire image. Moreover, storing information in terms of frequency takes much fewer bits than storing by pixel color, and no information is lost. When the image has to be displayed again, it is uncompressed into the original file. Below is an example of JPEG compression. This particular example models Discrete Cosine Transform (DCT), but its method and use are very similar to that of Fourier.

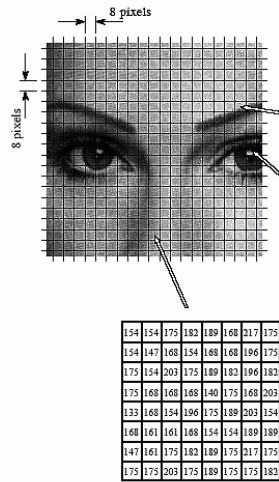


FIGURE 27-9
JPEG image division. JPEG transform compression starts by breaking the image into 8×8 groups, each containing 64 pixels. Three of these 8×8 groups are enlarged in this figure, showing the values of the individual pixels, a single byte value between 0 and 255.

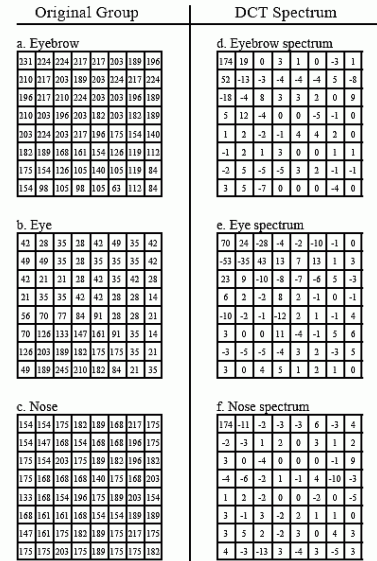
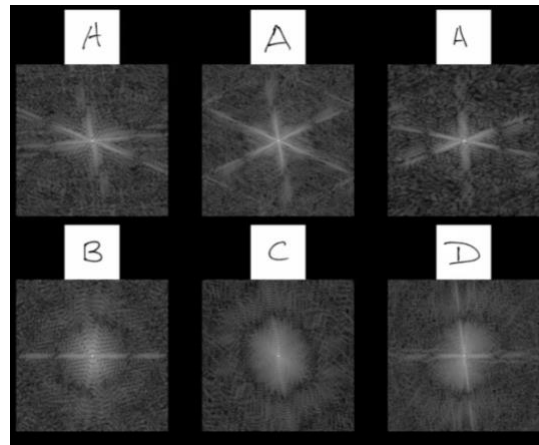


Figure. 5

It can be clearly seen that the transformation into the frequency spectrum reduced the amount of bits significantly. The need for data compression might not be apparent when using a picture as an example, but Fourier Transform is used with all different types of information. However, when taking into account HD video, made up of thousands of frames, the need for compression becomes more apparent. Fourier transform is also used when dealing with infinite streams of information, like radio, Wi-Fi, and video streaming, where this information has to be performed in real time. Imagine if every second of video were to take five seconds to broadcast, this would make live streaming impossible. FFT can do in fractions of a second what DFT would take several seconds to transform, and what would take hours and beyond to do without any form of transform or compression.

One final example is with image processing. While humans are very good and recognizing and categorizing images, computers have a hard time doing so. Even with recognizing letters, which should be a simple task, is extremely challenging for a machine. This would have clear advantages, like having postage categorization, license plate

identification, and hand-written form logging be automated. Transforming an image to the frequency domain creates patterns that are much more easily recognizable by a computer.



Fourier Transforms in Image Processing. Dr. Michelle Dunn,
Swinburne University of Technology

Figure. 6

While recognizing the above handwritten letters is very easy for a human, a computer does not see these letters as images, but rather a set of data. Finding patterns in that data that can span over many different types of handwriting and letters is extremely challenging, but transforming this data into the frequency domain creates patterns, like the ones displayed in the image, that are unique to each letter and more easily recognizable by the computer. Image compression and processing is just a small corner of the range of applications for Fourier Analysis, that span across Math, Physics, and Computer Sciences and make every day practices possible.

Fourier series draws a function or a set of data points into a periodic linear combination of sinusoids that maps the function from the time domain to the frequency domain. Fourier transform expands this domain into the infinite domain, making data representation in terms of frequency possible. Algorithm efficiency methods reduce the time of the algorithm to almost linear time, further increasing the efficiency and speed of

the transform. Thanks to Fourier, technologies like GPS, MP3, JPEG, Wi-Fi, and things beyond Computer Science, like multiplying large order polynomials, are made possible.

References

Morita, Kiyoshi. "Applied Fourier transform", 1901-
Tokyo, Japan : Ohmsha ; Burke, VA : IOS Press, c1995.

Book

Print

C.S Burrus. "Fast fourier transform and convolution algorithms: by Henri J. Nussbaumer", Laboratoire d'Informatique Technique, Swiss Federal Institute of Technology, Lausanne, Switzerland. Springer Series in Information Sciences, Vol. 2. Publishers: Springer-Verlag, Berlin/Heidelberg/New York, Heidelbergplatz 3, D-1000 Berlin 33, Fed. Rep. Germany, 1982 2nd ed., 273 pp.,

Signal Processing,

Volume 12, Issue 1,

1987,

(<https://www.sciencedirect.com/science/article/pii/0165168487900909>)

Steven W. Smith. "The Scientist and Engineer's Guide to Digital Signal Processing",
11/17/2001

<http://www.dspguide.com/pdfbook.htm>

Professor J. Daugman. Mathematical Methods for Computer Science. Part A: Fourier and related methods
<https://www.cl.cam.ac.uk/teaching/1213/MathMforCS/MathMforCSPartAslides.pdf>