Discrete Fourier Transform

Introduction

An understanding of what the discrete Fourier transform (DFT) is actually doing is discussed here. Many references exist that specify the mathematics, but it is not always clear what the mathematics actually mean. There are several different ways of understanding the fourier transform, this page will explain it in terms of correlation between a signal and sinusoids of various frequencies.

So what is the DFT? It is an algorithm that takes a signal and determines the 'frequency content' of the signal. For the following discussion, a 'signal' is any sequence of numbers, e.g. this is the first 12 numbers of a signal we will look at later (all samples here)

1.00, 0.62, -0.07, -0.87, -1.51, -1.81, -1.70, -1.24, -0.64, -0.15, 0.05, -0.10

We refer to these signals with x(n) where n is the index into the signal. x(0) = 1.00, x(1) = 0.62 etc. Signals such as this arise in many situations, for example all digital audio signals consist of sequences of numbers exactly like the one above. We will consider zero-mean signals, which means if you calculate the mean of the signal you get 0.

When we try to determine the 'frequency content', we are trying to decompose a complicated signal into simpler parts. Many signals are best described as a sum of many individual frequency components instead of time domain samples. The job of the Discrete Fourier Transform is to determine which frequencies a complicated signal is composed of.

Correlation

Correlation is a widely used concept in signal processing, which at its heart is a measure of how similar two signals are. If the correlation is high, then the signals are similar, if the correlation is near zero the signals are not very similar. In general, if you ever see an equation that looks like this:

$$\sum_{i=0}^{N} x(i)y(i)$$

where x and y are signals of some sort, you can say 'Aha! We are calculating the correlation (or similarity) between x and y!'. How does such a simple looking equation calculate the similarity between two signals? If x is very similar to y, then when x(i) is positive, y(i) will also be positive, and when x(i) is negative, y(i) will usually also be negative (since the signals are similar). If we multiply 2 positive numbers (x(i) and y(i)), we get another positive number. If we multiply two negatives, we get another positive number. Then when we sum up all these positive numbers we get a large positive number. What happens if the signals are not very similar? Then sometimes x(i) will be positive while y(i) is negative and vice versa. Other times x(i) will be positive and y(i) will also be positive. In this way when we calculate the final sum we add a few positive numbers and a few negative numbers, which results in a number near zero. This is the idea behind it. Very similar signals will achieve a high correlation, while very different signals will achieve a low correlation, and signals that are a little bit similar will get a correlation score somewhere in between.

Before we go further, I'll mention that correlation can also be negative. This happens when every time x(i) is positive, y(i) is negative and vice versa. This means the result will always be negative (because a negative times a positive is negative). A large negative correlation also means the signals are very similar, but one is inverted with respect to the other.

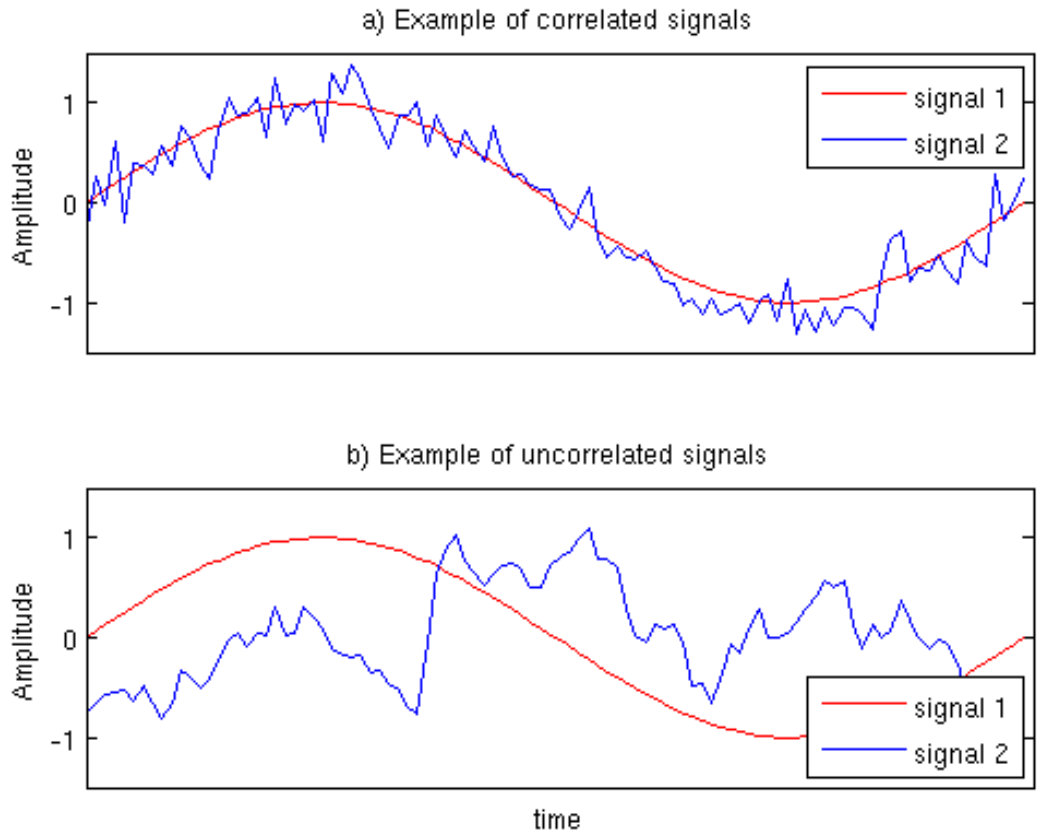

Figure 1: Example of correlated and uncorrelated signals

The Discrete Fourier Transform

How does Correlation help us understand the DFT? Have a look at the equation for the DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi kn/N}$$

Where we sweep k from 0 to N-1 to calculate all the DFT coefficients. When we say 'coefficient' we mean the values of X(k), so X(0) is the first coefficient, X(1) is the second etc. This equation should look very similar to the correlation equation we looked at earlier, because it is calculating the correlation between a signal $x(n)$ and a function $e^{-i2\pi kn/N}$. But what does it really mean for a signal to be similar to a complex exponential? It makes much more sense if we break up the exponential into sine and cosine components using the following identity:

$$e^{-i\theta} = \cos\theta - i\sin\theta$$

If we make $\theta = 2\pi kn/N$, our new DFT equation looks like this:

$$X(k) = \sum_{n=0}^{N-1} x(n) \left(\cos\left(2\pi kn/N\right) - i \sin\left(2\pi kn/N\right)\right)$$

With a little bit of algebra we can turn it into this:

$$X(k) = \boxed{\sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi kn}{N}\right)} - i \boxed{\sum_{n=0}^{N-1} x(n) \sin\left(\frac{2\pi kn}{N}\right)}$$

This looks much easier to tackle! It is in fact 2 correlation calculations (each correlation equation is in a blue rectangle), one with a sin wave and another with a cosine. First we calculate the correlation between our signal x(n) and a cosine of a certain frequency, and put the value we get into the real part of X(k). Then we do the same thing with a sine wave, and put the value we get into the imaginary component of X(k).

Correlation with a Cosine

The next question we wish to ask is how does determining the correlation between a cosine and our signal tells us anything useful? Consider a signal of length 100 samples that looks like the following (get the signal samples here)

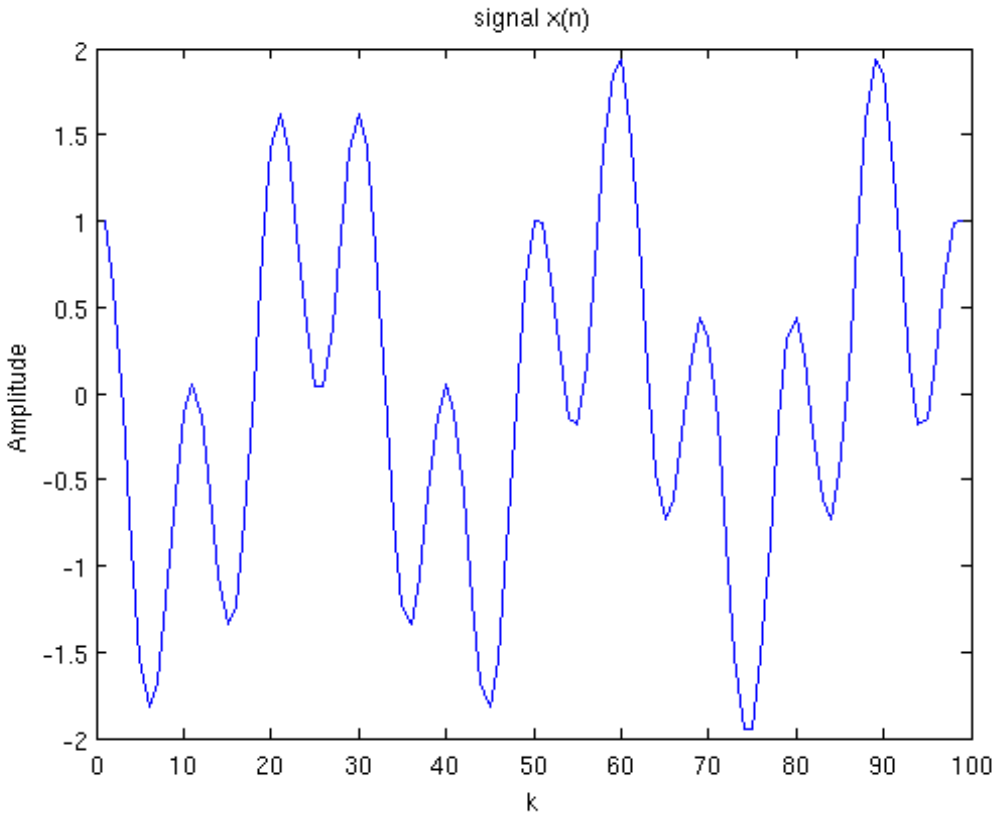

Figure 2: plot of example signal

We are going to go through the steps of computing how correlated this signal is with a sequence of cosine and sine waves of increasing frequency. This will be simulating the calculation of the real part of the DFT. Our first calculation will be for k=0 using the equation above:

$$X(0) = \sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi 0 n}{N}\right) + i \sum_{n=0}^{N-1} x(n) \sin\left(\frac{2\pi 0 n}{N}\right)$$
$$= \sum_{n=0}^{N-1} x(n) + i \sum_{n=0}^{N-1} 0$$

Here we calculated how correlated the signal is with another signal consisting only of ones (since cos(0) = 1). This turns out to be equal to the sum of the unmodified signal, which from the plot above we see should come out to be close to zero because half the signal looks to be above zero and half below. When we add all of it up the positive components will cancel with the negative components, resulting in a correlation equal to 1.

If we set k = 1 we get the following figure: plot(a) has $x(n)$ in blue, $\cos(2\pi 1n/N)$ in red and $-\sin(2\pi 1n/N)$ in black. Plot(b) is $x(n) * \cos(2\pi 1n/N)$, plot (c) is $x(n) * -\sin(2\pi 1n/N)$.

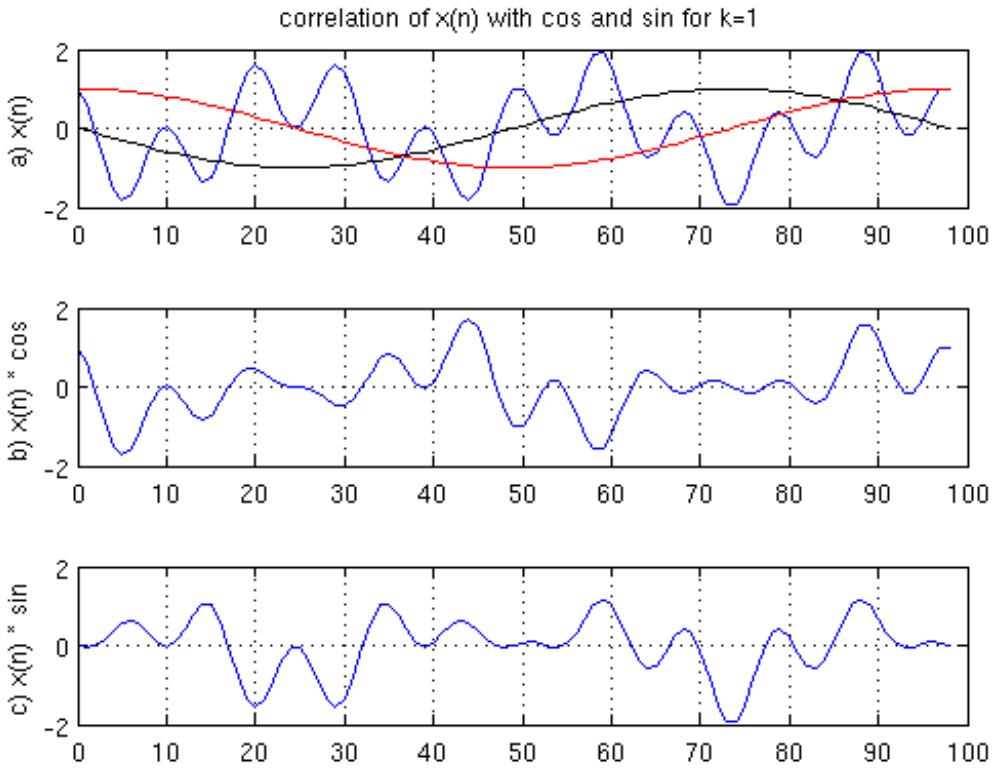

Figure 3: correlation between signal and sinusoids, k=1

The signals in (b) and (c) look like roughly half of the signal is below zero and half above, so when we add up the signals we expect a number near zero. Indeed $\sum_n x(n) * \cos(2\pi 1n/N)$ is equal to 1.0 and $\sum_n x(n) * -\sin(2\pi 1n/N)$ is equal to 0, so X(1) is 1+i0.

The next figure shows the same signals as the previous, except now k=3. Plot (a) shows x(n) along with the cos and sin waves. Plots (b) and (c) show the product of x(n) with the cos and sin signals respectively.

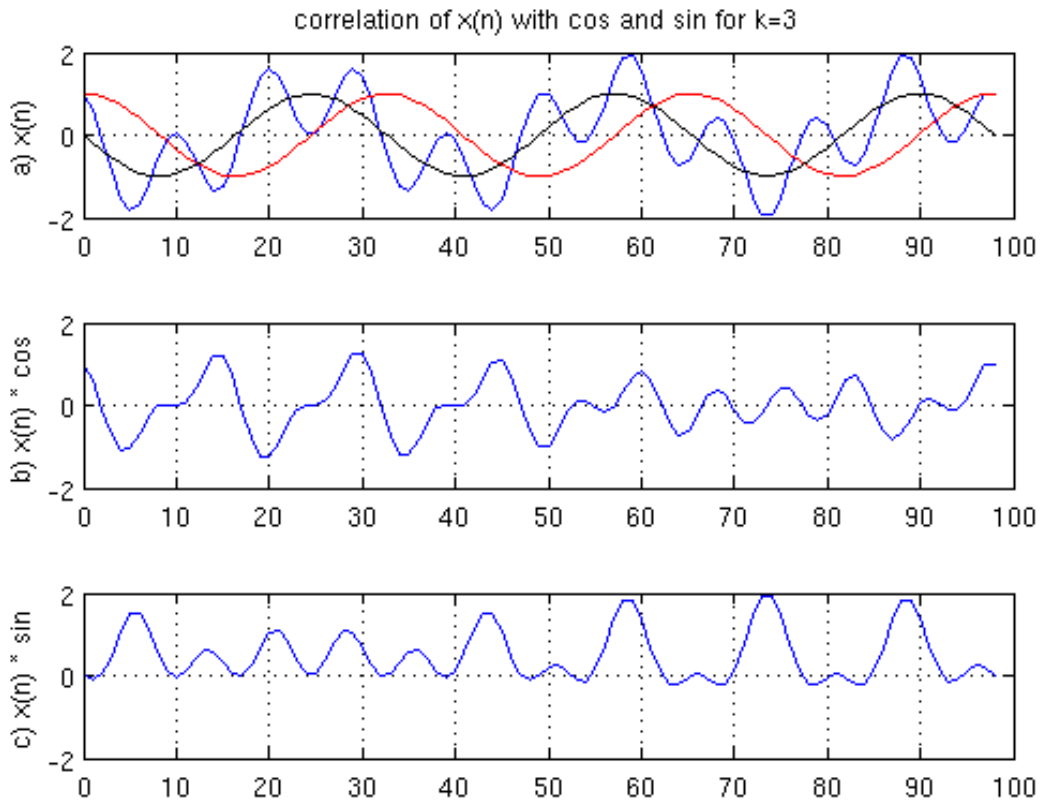

Figure 4: correlation between signal and sinusoids, k=3

It is evident that plot (c) is almost entirely positive, with very few negative parts. This means when we sum it up there will be almost no cancellation and we will get a large positive number. In this case the correlation with the cosine comes out to 0, while the correlation with the sin equals 49. This means the DFT coefficient for k = 3 (X(3)) is 0+i49. This process can be continued for each k until the complete DFT is obtained.

Our final discrete Fourier transform looks like this (real part on the left, imaginary part on the right):

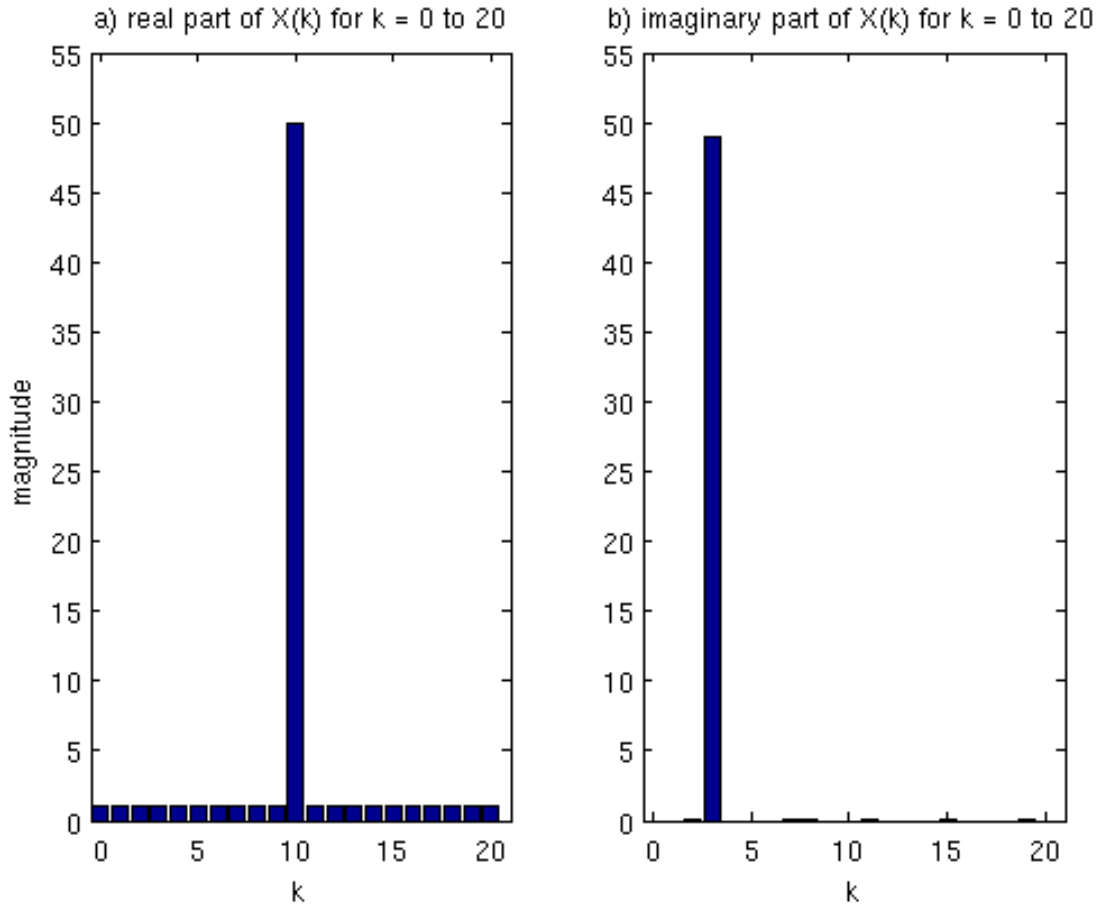

Figure 5: real (left) and imaginary (right) components of X(k)

We see that when k=10 we get a peak in the real part (the signal is similar to a cosine), and at k=3 we get a peak in the imaginary part (the signal is similar to a sine). In general though, we don't care if we have a cosine present or a sine, we just want to know if a sinusoid is present, irrespective of phase. To achieve this we combine the real and imaginary components to get the power spectrum:

$$P(k) = \mathrm{Re}(X(k))^2 + \mathrm{Im}(X(k))^2$$

Note that the power spectrum P(k) has no imaginary component. If we calculate the power spectrum from X(k), we get the following picture:

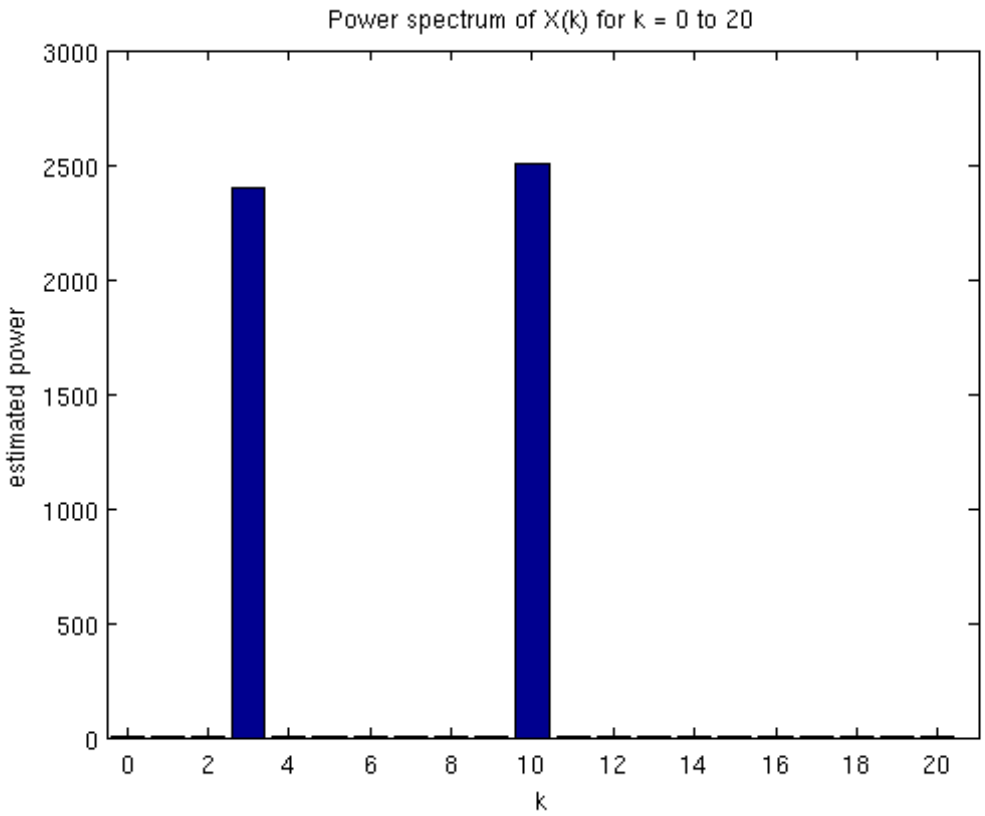

Figure 6: Power spectrum of X(k)

Here we can clearly see that 2 frequencies are present in our signal and little else, which is what we set out to determine.

To convert these values of k into actual frequencies (measured in Hertz), we need the sampling rate of our original signal (fs) and the following formula:

$$f = \frac{k \times \mathrm{fs}}{N}$$

Where fs is the sampling frequency and N is the number of samples in our signal (100 for the example above).

Wrapping up

In an attempt to simplify the discussion we have ignored a few details which we will explain here. We have assumed that all our signals are zero mean, so that the correlation between them is zero when they are dissimilar. This is not necessarily true, but high correlations always mean the signals are similar and lower correlations dissimilar. It is in any case easy enough to subtract the mean from any signals we wish to consider.

In the example above, we calculated the DFT for k = 0 to 20. If we kept calculating coefficients for higher k, we would find that the power spectrum is reflected around N/2. The real part is an even function of k, while the imaginary part is an odd function of k.

Discrete cosine transform

A **discrete cosine transform** (**DCT**) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The DCT, first proposed by Nasir Ahmed in 1972, is a widely used transformation technique in signal processing and data compression. It is used in most digital media, including digital images (such as JPEG and HEIF, where small high-frequency components can be discarded), digital video (such as MPEG and H.26x), digital audio (such as Dolby Digital, MP3 and AAC), digital television (such as SDTV, HDTV and VOD), digital radio (such as AAC+ and DAB+), and speech coding (such as AAC-LD, Siren and Opus). DCTs are also important to numerous other applications in science and engineering, such as digital signal processing, telecommunication devices, reducing network bandwidth usage, and spectral methods for the numerical solution of partial differential equations.

The use of cosine rather than sine functions is critical for compression, since it turns out (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions. In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. The DCTs are generally related to Fourier series coefficients of a periodically and symmetrically extended sequence whereas DFTs are related to Fourier series coefficients of only periodically extended sequences. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), whereas in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common.

The most common variant of discrete cosine transform is the type-II DCT, which is often called simply "the DCT". This was the original DCT as first proposed by Ahmed. Its inverse, the type-III DCT, is correspondingly often called simply "the inverse DCT" or "the IDCT". Two related transforms are the discrete sine transform (DST), which is equivalent to a DFT of real and *odd* functions, and the modified discrete cosine transform (MDCT), which is based on a DCT of *overlapping* data. Multidimensional DCTs (MD DCTs) are developed to extend the concept of DCT to MD signals. There are several algorithms to compute MD DCT. A variety of fast algorithms have been developed to reduce the computational complexity of implementing DCT. One of these is the integer DCT (IntDCT), an integer approximation of the standard DCT, used in several ISO/IEC and ITU-T international standards.

DCT compression, also known as block compression, compresses data in sets of discrete DCT blocks. DCT blocks can have a number of sizes, including 8x8 pixels for the standard DCT, and varied integer DCT sizes between 4x4 and 32x32 pixels. The DCT has a strong "energy compaction" property, capable of achieving high quality at high data compression ratios However, blocky compression artifacts can appear when heavy DCT compression is applied.

Applications

The DCT is the most widely used transformation technique in signal processing, and by far the most widely used linear transform in data compression. Uncompressed digital media as well as lossless compression had impractically high memory and bandwidth requirements, which was significantly reduced by the highly efficient DCT lossy compression technique, capable of achieving data compression ratios from 8:1 to 14:1 for near-studio-quality, up to 100:1 for acceptable-quality content. DCT compression standards are used in digital media technologies, such as digital images, digital photos, digital

video, streaming media, digital television, streaming television, video-on-demand (VOD), digital cinema, high-definition video (HD video), and high-definition television (HDTV).

The DCT, and in particular the DCT-II, is often used in signal and image processing, especially for lossy compression, because it has a strong "energy compaction" property: in typical applications, most of the signal information tends to be concentrated in a few low-frequency components of the DCT. For strongly correlated Markov processes, the DCT can approach the compaction efficiency of the Karhunen-Loève transform (which is optimal in the decorrelation sense). As explained below, this stems from the boundary conditions implicit in the cosine functions.

DCTs are also widely employed in solving partial differential equations by spectral methods, where the different variants of the DCT correspond to slightly different even/odd boundary conditions at the two ends of the array.

DCTs are also closely related to Chebyshev polynomials, and fast DCT algorithms (below) are used in Chebyshev approximation of arbitrary functions by series of Chebyshev polynomials, for example in Clenshaw–Curtis quadrature.

The DCT is the coding standard for multimedia telecommunication devices. It is widely used for bit rate reduction, and reducing network bandwidth usage. DCT compression significantly reduces the amount of memory and bandwidth required for digital signals.

Discrete sine transform

In mathematics, the **discrete sine transform (DST)** is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using a purely real matrix. It is equivalent to the imaginary parts of a DFT of roughly twice the length, operating on real data with odd symmetry (since the Fourier transform of a real and odd function is imaginary and odd), where in some variants the input and/or output data are shifted by half a sample.

A family of transforms composed of **sine** and **sine hyperbolic** functions exists. These transforms are made based on the ***natural vibration*** of thin square plates with different boundary conditions.

The DST is related to the discrete cosine transform (DCT), which is equivalent to a DFT of real and *even* functions. See the DCT article for a general discussion of how the boundary conditions relate the various DCT and DST types. Generally, the DST is derived from the DCT by replacing the Neumann condition at $x=0$ with a Dirichlet condition. Both the DCT and the DST were described by Nasir Ahmed T. Natarajan and K.R. Rao in 1974. The type-I DST (DST-I) was later described by Anil K. Jain in 1976, and the type-II DST (DST-II) was then described by H.B. Kekra and J.K. Solanka in 1978.

Applications

DSTs are widely employed in solving partial differential equations by spectral methods, where the different variants of the DST correspond to slightly different odd/even boundary conditions at the two ends of the array.

Informal overview

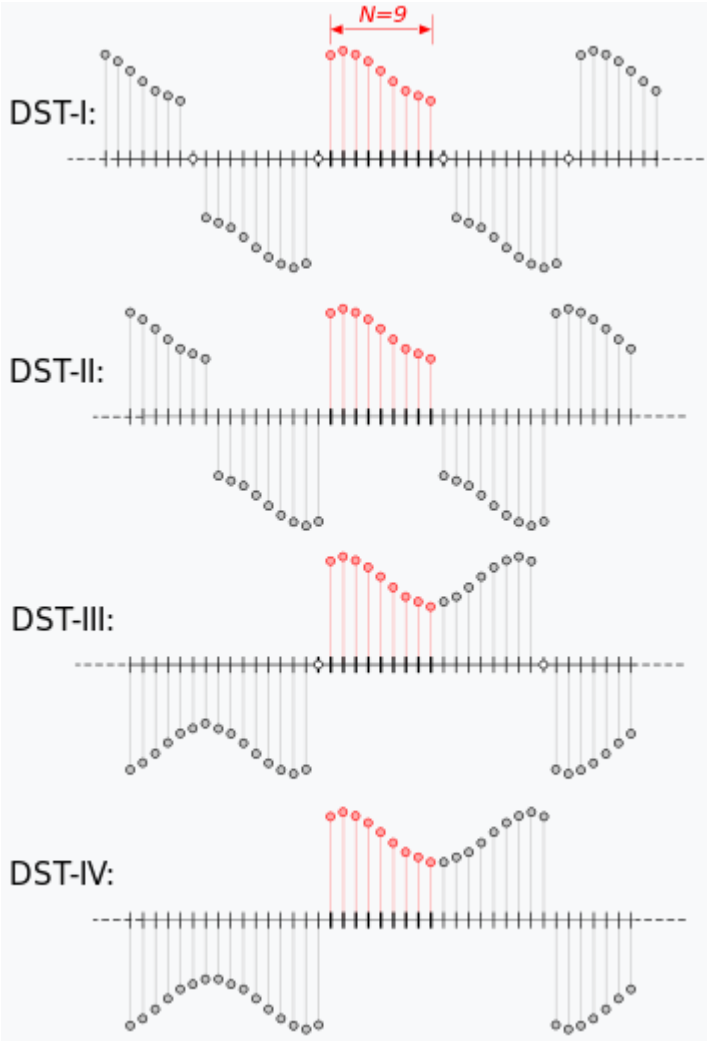

Illustration of the implicit even/odd extensions of DST input data, for N=9 data points (red dots), for the four most common types of DST (types I–IV).

Like any Fourier-related transform, discrete sine transforms (DSTs) express a function or a signal in terms of a sum of sinusoids with different frequencies and amplitudes. Like the discrete Fourier transform (DFT), a DST operates on a function at a finite number of discrete data points. The obvious distinction between a DST and a DFT is that the former uses only sine functions, while the latter uses both cosines and sines (in the form of complex exponentials). However, this visible difference is merely a consequence of a deeper distinction: a DST implies different boundary conditions than the DFT or other related transforms.

The Fourier-related transforms that operate on a function over a finite domain, such as the DFT or DST or a Fourier series, can be thought of as implicitly defining an *extension* of that function outside the domain. That is, once you write a function as a sum of sinusoids, you can evaluate that sum at any , even for where the original was not specified. The DFT, like the Fourier series, implies a periodic extension of the original function. A DST, like a sine transform, implies an odd extension of the original function.

However, because DSTs operate on *finite, discrete* sequences, two issues arise that do not apply for the continuous sine transform. First, one has to specify whether the function is even or odd at *both* the left and right boundaries of the domain (i.e. the min-*n* and max-*n* boundaries in the definitions below, respectively). Second, one has to specify around *what point* the function is even or odd. In particular, consider a sequence (*a*,*b*,*c*) of three equally spaced data points, and say that we specify an odd *left* boundary. There are two sensible possibilities: either the data is odd about the point *prior* to *a*, in which case the odd extension is (−*c*,−*b*,−*a*,0,*a*,*b*,*c*), or the data is odd about the point *halfway* between *a* and the previous point, in which case the odd extension is (−*c*,−*b*,−*a*,*a*,*b*,*c*)

These choices lead to all the standard variations of DSTs and also discrete cosine transforms (DCTs). Each boundary can be either even or odd (2 choices per boundary) and can be symmetric about a data point or the point halfway between two data points (2 choices

per boundary), for a total of possibilities. Half of these possibilities, those where the *left* boundary is odd, correspond to the 8 types of DST; the other half are the 8 types of DCT.

These different boundary conditions strongly affect the applications of the transform, and lead to uniquely useful properties for the various DCT types. Most directly, when using Fourier-related transforms to solve partial differential equations by spectral methods, the boundary conditions are directly specified as a part of the problem being solved.

Definition

Formally, the discrete sine transform is a linear, invertible function $F : \mathbf{R}^N \to \mathbf{R}^N$ (where $\mathbf{R}$ denotes the set of real numbers), or equivalently an $N \times N$ square matrix. There are several variants of the DST with slightly modified definitions. The N real numbers x_0, x_{N-1} are transformed into the N real numbers X_0, X_{N-1} according to one of the formulas:

DST-I

$$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N+1}(n+1)(k+1)\right] \qquad k = 0, \ldots, N-1$$

The DST-I matrix is orthogonal (up to a scale factor).

A DST-I is exactly equivalent to a DFT of a real sequence that is odd around the zero-th and middle points, scaled by 1/2. For example, a DST-I of N=3 real numbers (*a*,*b*,*c*) is exactly equivalent to a DFT of eight real numbers (0,*a*,*b*,*c*,0,−*c*,−*b*,−*a*) (odd symmetry), scaled by 1/2. (In contrast, DST types II–IV involve a half-sample shift in the equivalent DFT.) This is the reason for the $N + 1$ in the denominator of the sine function: the equivalent DFT has $2(N+1)$ points and has $2\pi/2(N+1)$ in its sinusoid frequency, so the DST-I has $\pi/(N+1)$ in its frequency.

Thus, the DST-I corresponds to the boundary conditions: x_n is odd around $n = -1$ and odd around $n=N$; similarly for X_k.

DST-II

$$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)(k+1)\right] \qquad k = 0, \ldots, N-1$$

Some authors further multiply the X_{N-1} term by $1/\sqrt{2}$ (see below for the corresponding change in DST-III). This makes the DST-II matrix orthogonal (up to a scale factor), but breaks the direct correspondence with a real-odd DFT of half-shifted input.

The DST-II implies the boundary conditions: x_n is odd around $n = -1/2$ and odd around $n = N - 1/2$; X_k is odd around $k = -1$ and even around $k = N - 1$.

DST-III

$$X_k = \frac{(-1)^k}{2} x_{N-1} + \sum_{n=0}^{N-2} x_n \sin\left[\frac{\pi}{N}(n+1)\left(k+\frac{1}{2}\right)\right] \qquad k = 0, \ldots, N-1$$

Some authors further multiply the x_{N-1} term by $\sqrt{2}$ (see above for the corresponding change in DST-II). This makes the DST-III matrix orthogonal (up to a scale factor), but breaks the direct correspondence with a real-odd DFT of half-shifted output.

The DST-III implies the boundary conditions: x_n is odd around $n = -1$ and even around $n = N - 1$; X_k is odd around $k = -1/2$ and odd around $k = N - 1/2$.

DST-IV

$$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right] \qquad k = 0, \ldots, N-1$$

The DST-IV matrix is orthogonal (up to a scale factor).

The DST-IV implies the boundary conditions: x_n is odd around $n = -1/2$ and even around $n = N - 1/2$; similarly for X_k.

DST V–VIII

DST types I–IV are equivalent to real-odd DFTs of even order. In principle, there are actually four additional types of discrete sine transform (Martucci, 1994), corresponding to real-odd DFTs of logically odd order, which have factors of N+1/2 in the denominators of the sine arguments. However, these variants seem to be rarely used in practice.

Inverse transforms

The inverse of DST-I is DST-I multiplied by $2/(N + 1)$. The inverse of DST-IV is DST-IV multiplied by $2/N$. The inverse of DST-II is DST-III multiplied by $2/N$ (and vice versa).

As for the DFT, the normalization factor in front of these transform definitions is merely a convention and differs between treatments. For example, some authors multiply the transforms by $\sqrt{2/N}$ so that the inverse does not require any additional multiplicative factor.

Computation

Although the direct application of these formulas would require $O(N^2)$ operations, it is possible to compute the same thing with only $O(N \log N)$ complexity by factorizing the computation similar to the fast Fourier transform (FFT). (One can also compute DSTs via FFTs combined with $O(N)$ pre- and post-processing steps.)

A DST-III or DST-IV can be computed from a DCT-III or DCT-IV (see discrete cosine transform), respectively, by reversing the order of the inputs and flipping the sign of every other output, and vice versa for DST-II from DCT-II. In this way it follows that types II–IV of the DST require exactly the same number of arithmetic operations (additions and multiplications) as the corresponding DCT types.