Click Here

Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. In digital signal processing, a discrete Fourier series (DFS) is a Fourier series whose sinusoidal components are functions of a discrete variable instead of a continuous variable. The result of the series is also a function of the discrete variable, i.e. a discrete sequence. A Fourier series, by nature, has a discrete set of components with a discrete set of coefficients, also a discrete sequence. So a DFS is a representation of one sequence in terms of another sequence. Well known examples are the Discrete Fourier transform and its inverse transform.[1]:ch 8.1 The exponential form of Fourier series is given by: s (t) = ∑ k = − ∞ ∞ S [k] ⋅ e i 2 π k P t , {\displaystyle s(t)=\sum _{k=-\infty }^{\infty }S[k]\cdot e^{i2\pi {\frac {k}{P}}t},} which is periodic with an arbitrary period denoted by P . {\displaystyle P.} When continuous time t {\displaystyle t} is replaced by discrete time n T , {\displaystyle nT,} for integer values of n {\displaystyle n} and time interval T , {\displaystyle T,} the series becomes: s (n T) = ∑ k = − ∞ ∞ S [k] ⋅ e i 2 π k P n T , n ∈ Z . {\displaystyle s(nT)=\sum _{k=-\infty }^{\infty }S[k]\cdot e^{i2\pi {\frac {k}{P}}nT},\quad n\in \mathbb {Z} .} With n {\displaystyle n} constrained to integer values, we normally constrain the ratio P / T = N {\displaystyle P/T=N} to an integer value, resulting in an N {\displaystyle N} -periodic function: Discrete Fourier series s N [n] ≜ s (n T) = ∑ k = − ∞ ∞ S [k] ⋅ e i 2 π k n N {\displaystyle s_{_{N}}[n]\triangleq s(nT)=\sum _{k=-\infty }^{\infty }S[k]\cdot e^{i2\pi {\frac {k}{N}}n}} which are harmonics of a fundamental digital frequency 1 / N . {\displaystyle 1/N.} The N {\displaystyle N} subscript reminds us of its periodicity. And we note that some authors will refer to just the S [k] {\displaystyle S[k]} coefficients themselves as a discrete Fourier series.[2]:p.85 (eq 15a) Due to the N {\displaystyle N} -periodicity of the e i 2 π k N n {\displaystyle e^{i2\pi {\tfrac {k}{N}}n}} kernel, the infinite summation can be "folded" as follows: s N [n] = ∑ m = − ∞ ∞ (∑ k = 0 N − 1 e i 2 π k − m N N n S [k − m N]) = ∑ k = 0 N − 1 e i 2 π k N n (∑ m = − ∞ ∞ S [k − m N]) ⏟ ≜ S N [k] , {\displaystyle {\begin{aligned}s_{_{N}}[n]&=\sum _{m=-\infty }^{\infty }\left(\sum _{k=0}^{N-1}e^{i2\pi {\tfrac {k-mN}{N}}n}\ S[k-mN]\right)\\&=\sum _{k=0}^{N-1}e^{i2\pi {\tfrac {k}{N}}n}\underbrace {\left(\sum _{m=-\infty }^{\infty }S[k-mN]\right)} _{\triangleq S_{N}[k]},\end{aligned}}} which is the inverse DFT of one cycle of the periodic summation, S N . {\displaystyle S_{N}.} [1]:p.542 (eq 8.4) [3]:p.77 (eq 4.24) ^ a b Oppenheim, Alan V.; Schafer, Ronald W.; Buck, John R. (1999). Discrete-time signal processing (2nd ed.). Upper Saddle River, N.J.: Prentice Hall. ISBN 0-13-754920-2. samples of the Fourier transform of an aperiodic sequence x[n] can be thought of as DFS coefficients of a periodic sequence obtained through summing periodic replicas of x[n]. ... The Fourier series coefficients can be interpreted as a sequence of finite length for k=0,...,(N-1), and zero otherwise, or as a periodic sequence defined for all k. ^ Nuttall, Albert H. (Feb 1981). "Some Windows with Very Good Sidelobe Behavior". IEEE Transactions on Acoustics, Speech, and Signal Processing. 29 (1): 84–91. doi:10.1109/TASSP.1981.1163506. ^ Prandoni, Paolo; Vetterli, Martin (2008). Signal Processing for Communications (PDF) (1 ed.). Boca Raton,FL: CRC Press. pp. 72, 76. ISBN 978-1-4200-7046-0. Retrieved 4 October 2020. the DFS coefficients for the periodized signal are a discrete set of values for its DTFT Retrieved from " Signal Processing \(ewcommand{\sumN}[1]{\sum_{#1\in\langle N\rangle}}\) As computer scientists we most often work with digital signals, i.e. DT signals. For infinite length (non periodic) signals the DT Fourier transform from the previous chapter is more of a theoretical tool to analyze DT signals and DT filters. Actually calculating the DT Fourier transform for a signal in practice is impossible as an infinite length signal cannot be stored in computer memory. let alone be processed in its entirety to calculate its Fourier transform. In practice only a small part of a discrete signal is used to analyze its frequency components. That is where the discrete time Fourier series comes in. We start with a signal \(x[n]\) that is only defined from \(n=0\) to \(n=N-1\). This signal can be seen as one period of an infinite length signal with period \(N\) making the DTFS the discrete analog of the CT Fourier series. The convolution kernels \(h[n]\) that are used are often defined analytically (i.e. we have a formula for it) and in many cases we are capable to calculate its discrete time Fourier transform. But for observed signals, that are defined on the infinite time axis, calculation of the Fourier transform is not feasible. The DTFS is the transform for frequency analysis to be used in practice. It is also known as the DFT (Discrete Fourier Transform not to be confused with the Discrete Time Fourier Transform) and an efficient algorithm for the DFT is the FFT (Fast Fourier Transform). Unfortunately the choice for the normalization constants (in both the Fourier transform and its inverse) may differ. When reading articles or when using software implementing Fourier transforms it is always wise to look up the definitions that are being used. We now have a DT signal that is periodic with period \(N\), i.e. \(x[n+N]=x[n]\). In practice given only \(N\) samples of a DT signal we just assume that we know is one period of a periodic signal. The synthesis equation is: \[x[n] = \sum_{k=0}^{N-1} a_k e^{jk\frac{2\pi}{N}n}\] where \[\Omega_0 = \frac{2\pi}{N}\] is the fundamental frequency of the signal \(x[n]\) (of length \(N\)). Below we sketch a DT periodic signal with \(N=8\). The 8 basis signals are the complex exponentials: \[\phi_k[n] = e^{jk\frac{2\pi}{N}n} = e^{jk\Omega_0n}\] Show code for figure 1import numpy as np 2import matplotlib.pyplot as plt 3 4def phi(n, k): 5 N = len(n) 6 Omega_0 = 2*np.pi / N 7 return np.exp(1j*k*Omega_0*n) 8 9N = 8 10n = np.arange(N) 11fig, axs = plt.subplots(nrows=N, figsize=(6,10), sharex=True) 12fig.tight_layout() 13 14for k in range(N): 15 axs[k].stem(n, phi(n, k).real, use_line_collection=True) 16 axs[k].set_ylim(-1.5,1.5) 17 axs[k].set_xlim(-3,N) 18 axs[k].text(-2,0,'k = %d' % k) # why don't i see them...??? 19 axs[k].axis('off') 20 21plt.savefig('source/figures/discrete_harmonics.png') Fig. 3.9 Discrete Harmonics. Shown are the real parts of the complex exponentials \(\phi_k[n] = e^{jk\Omega_0n}\) for \(k=0,\ldots,7\).¶ Observe that for \(k=0,1,2,3,4\) the frequency is increasing from zero to the maximal frequency exponential \(e^{j\pi n}\) where the value \(x[n]\) jumps from +1 to -1 and back for every \(n\). The analysis equation in this case is: \[a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j k \frac{2\pi}{N} n}\] For the discrete time Fourier series it is not too hard to derive the analysis equation given the synthesis equation. Let’s start with the synthesis equation and multiply each side with a complex exponential function \[x[n] e^{-j k' \frac{2\pi}{N} n} = \sum_{k=0}^{N-1} a_k e^{jk\frac{2\pi}{N}n} e^{-j k' \frac{2\pi}{N} n}\] Now we take the sum over all \(n\) on both sides: \[\begin{split}\sum_{n=0}^{N-1} x[n] e^{-j k' \frac{2\pi}{N} n} &= \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} a_k e^{jk\frac{2\pi}{N}n} e^{-j k' \frac{2\pi}{N} n}\\ &= \sum_{k=0}^{N-1} a_k \sum_{n=0}^{N-1} e^{j(k-k')\frac{2\pi}{N}n} \\ &= \sum_{k=0}^{N-1} a_k N \delta[k-k']\\ &= N a_{k'}\end{split}\] and thus: \[a_{k'} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j k' \frac{2\pi}{N} n}\] In this derivation we have used: \[\sum_{n=0}^{N-1} e^{j m \frac{2\pi}{N} n} = N\delta[m]\] Convince yourself that this is true (easily done by plotting the real and imaginary parts of the complex exponential function. Time Domain Frequency Domain Synthesis (Inverse Fourier Transform) \[x[n] = \sumN{k} a_k e^{jk \frac{2\pi}{N}n}\] Analysis (Fourier Transform) \[a_k = \frac{1}{N} \sumN{n} x[n] e^{-j k \frac{2\pi}{N} n}\] Real Signals Symmetry in Fourier transform Even Signals Real Fourier transform Odd Signals Imaginary Fourier transform \[a_k \text{ is imaginary}\] Difference \[(1-e^{-j \frac{2\pi}{N} k})a_k\] Time Shift Phase factor \[e^{-j k \frac{1\pi}{N} n_0} a_k\] Convolution \[x[n] \ast_N y[n] = \sumN{k} x[k] y[n-k]\] Multiplication Multiplication Convolution Real SignalsLet \(x[n]\) be a real valued signal. The analysis equation for \(a_{-k}\) is: \[a_{-k} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{jk\frac{2\pi}{N} n}\] and because \(x[n]\) is real we have that \[a_{-k}^\ast = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N} n} = a_k\] SymmetryNote that the complex exponentials \(e^{j k \frac{2\pi}{N} n\) are periodic with period \(N\) for all \(k\) but then a summation of these exponentials is periodic as well and thus or \[a_{N-k} = a_{-k} = a_k^\star\] For even \(N\) we have \[\begin{split}a_0 &\mbox{ is real}\\ a_1 &= a_{N-1}^\star\\ a_2 &= a_{N-2}^\star\\ \vdots\\ a_{N/2} &\mbox{ is real}\\ \vdots\\ a_{N-2} &= a_2^\star\\ a_{N-1} &= a_1^\star\end{split}\] Synthesis of Real Valued FunctionThe synthesis equation for \(N\) even is \[\begin{split}x[n] &= \sum_{k=0}^{N-1} a_k e^{j k \frac{2\pi}{N} n}\\ &= a_0 + \sum_{k=1}^{N/2-1} a_k e^{j k \frac{2\pi}{N} n} + a_{N/2} e^{j\pi n} + \sum_{k=N/2+1}^{N-1} a_k e^{j k \frac{2\pi}{N} n}\\ &= a_0 + \sum_{k=1}^{N/2-1} a_k e^{j k \frac{2\pi}{N} n} + a_{N/2} e^{j\pi n} + \sum_{k=1}^{N/2-1} a_{N-k} e^{j (N-k) \frac{2\pi}{N} n}\\ &= a_0 + \sum_{k=1}^{N/2-1}\left(a_k e^{j k \frac{2\pi}{N} n} + a_{N-k} e^{j (N-k) \frac{2\pi}{N} n}\right) + a_{N/2} e^{j\pi n}\\ &= a_0 + \sum_{k=1}^{N/2-1}\left(a_k e^{j k \frac{2\pi}{N} n} + a_k^\star e^{-j k \frac{2\pi}{N} n}\right) + a_{N/2} e^{j\pi n}\end{split}\] We now write \[a_k = |a_k| e^{j\angle a_k}\text{ and thus } a_k^\star = |a_k| e^{-j\angle a_k}\] Leading to \[\begin{split}x[n] &= a_0 + \sum_{k=1}^{N/2-1}|a_k|\left(e^{j (k \frac{2\pi}{N} n + \angle a_k)} + e^{-j (k \frac{2\pi}{N} n + \angle a_k)} \right) + a_{N/2} e^{j\pi n}\\ &= a_0 + 2 \sum_{k=1}^{N/2-1}|a_k| \cos\left(k \frac{2\pi}{N} n + \angle a_k\right) + a_{N/2} \cos(\pi n)\end{split}\] Both time and frequency are discrete for the DT Fourier Series. Consider the \(k\)-th frequency component corresponding with complex exponential \[\phi_k[n] = e^{j k \frac{2\pi}{N} n}\] The radial frequency of this signal is \[\Omega = k\frac{2\pi}{N}\] We assume \(N\) is even and thus for \(k=N/2\) we have: \[k=N/2: \quad \Omega = \pi\] corresponding with maximal radial frequency that can be represented with the given sampling. Like in a previous section we can link the radial frequency with the time domain radial frequency \(\w\) and frequency \(f\): \[w = \frac{\Omega}{T_s}, \quad f = \frac{\Omega}{2\pi T_s} = \frac{k}{NT_s}\] where \(T_s\) is the sample time. The relation between \(n\) and time is simply given as \(t=nT_s\). Below a plot of the DT function \[x[n] = 5\sin(2\pi n / N) + \sin(14\pi n / N), \quad n=0,1,\ldots,63\] and its DT Fourier Transform. Please make sure you understand the plot. You should also be able to plot time on the horizontal axis in the first plot and frequency (in Hz) in the second plot. Show code for figure1plt.clf() 2 3from DTP import example_timefreq 4 5example_timefreq() 6plt.savefig('source/figures/example_timefreq.png') Fig. 3.10 The function \(x[n]\) and its discrete Fourier series.¶ Any signal in practice is a discrete time signal but furthermore it is a signal for a finite number of samples. No wonder that the computational way to do frequency analysis is by using the DT Fourier Series. Unfortunately to add to the confusion the DTFS is known as the Discrete Fourier Transform (DFT) and the fast algorithm to calculate the DFT is called the FFT (Fast Fourier Transform). And even more, the normalization factors are chosen differently than what we used in this section. In numpy.fft we find that the analysis equation (in the default setting) is: \[A_k = \sum_{m=0}^{n-1} a_m e^{-2\pi j \frac{mk}{n}}\] Note the change of notation. The \(A_k\)’s are the Fourier coefficients whereas the \(a_k\)’s are the samples of the signal. Also note that there is no scaling factor in the analysis equation and the \(n\) is the number of samples both in time as well as frequency domain. The synthesis equation is the inverse DFT: \[a_m = \frac{1}{n} \sum_{k=0}^{n-1} A_k e^{2\pi j \frac{mk}{n}}\] Again carefully note the differences with the DT Fourier Series. In the default setting of the FFT in numpy (i.e. parameter norm = ‘backward’) the scaling factors are opposite of what we have used in this chapter (i.e. we have used factor \(1/N\) in the analysis equation and factor \(1\) in the synthesis equation). As an example let us start with a simple block function Fig. 3.11 Block function and its discrete Fourier series.¶ The python code is given below. def example_DFT_block(): N = 32 n = np.arange(32) x = np.zeros_like(n) x[:4] = 1 x[-3:] = 1 fig, axs = plt.subplots(3) axs[0].stem(n, x) a = fft(x)/N axs[1].stem(n, a.real) axs[2].stem(n-N//2, np.fft.fftshift(a).real) print(n-N//2) Todo Code in rst file zetten! Maak de import van fft zichtbaar Look again at the analysis equation. It takes \(N\) numbers \(x[0],\ldots,x[N-1]\) and produces \(N\) complex numbers \(a_0,\ldots,a_{N-1}\). And it does so in a linear fashion meaning that it can be represented as a matrix operation. Let \[\begin{split}\v x &= \begin{pmatrix}x[0] & x[1] & \cdots & x[N-1]\end{pmatrix}\T\\ \v a &= \begin{pmatrix}a_0 & a_1 & \cdots & a_{N-1}\end{pmatrix}\T\\ W_N &= e^{-j \frac{2\pi}{N}}\\ F_{kn} &= e^{-j k \frac{2\pi}{N} n} = W_N^{kn}\end{split}\] then: \[\v a = \frac{1}{N} F_N \v x\] The matrix \(F_N\) is called the Fourier matrix. First we start with a simple straightforward implementation of the DFT: \[A_k = \sum_{m=0}^{N-1} a_m e^{-2\pi j \frac{mk}{N}}\] Given the signal \(a_m\) (we will follow the numpy fft documentation here) as the vector \(\v a\) we may calculate the DFT coefficients \(A_k\) (in vector \(\v A\)) as: where \(F\) is the Fourier matrix with elements \(F_{kn}=W_N^{kn}\). In Python/Numpy the DFT is easy to implement: def DFT(a): """Calculate the DFT of signal `a`""" N = len(a) n = arange(N) W = exp(-1j*2*pi/N) F = W**outer(n,n) return F @ a This is indeed what the fft from numpy calculates as well: 1from DFT_FFT import DFT 2from numpy.fft import fft 3 4a = np.random.random(512) 5A1 = DFT(a) 6A2 = fft(a) 7print(np.allclose(A1,A2)) To come up with a faster algorithm for the DFT we rewrite the analysis equation. We assume that \(N\) is some power of \(2\). First we split the sum over the inputs into the even and odd elements: \[\begin{split}A_k &= \sum_{m=0}^{N-1} a_m e^{-2\pi j \frac{mk}{N}}\\ &= \sum_{m=0}^{N/2-1} a_{2m} e^{-2\pi j \frac{2mk}{N}} + \sum_{m=0}^{N/2-1} a_{2m+1} e^{-2\pi j \frac{(2m+1)k}{N}}\\ &= \sum_{m=0}^{N/2-1} a_{2m} e^{-2\pi j \frac{mk}{N/2}} + e^{-2\pi j k/N} \sum_{m=0}^{N/2-1} a_{2m+1} e^{-2\pi j\frac{mk}{N/2}}\end{split}\] Note that in the summations over \(m\) runs from \(0\) to \(N/2\), but \(k\) still runs from \(0\) to \(N\). But also note that the exponential \(e^{-2\pi j mk / (N/2)}\) considered as function of \(k\) is periodic with period \(N/2\). So altough both summations in the last equation have to be done for \(k=0,\ldots,N\) we only have to calculate the vasummations for \(k=0,\ldots,N/2\) and repeat these values for larger \(k\)’s. Note that for \(k=0,\ldots,N/2\) the the \(N/2\) values of the first summation are the DFT of the even elements of the input signal and the values in the second summation are the DFT of the odd elements of the input signal. Carefully note that the phase factor \(e^{-2\pi j k/N}\) must be calculated for \(k=0,\ldots,N\). So we can decompose a DFT of \(N\) samples into two DFT’s of each \(N/2\) samples. But of course a DFT of N/2 samples can be decomposed into two DFT’s of \(N/5\) samples. This leads to the following recursive algorithm: def FFT(a): """ Calculate the DFT using the FFT (in recursive form) Parameters ---------- a : numpy array (n,) the signal, len(a) should be power of 2 """ N = len(a) if N % 2 > 0: raise ValueError("Size of a must be a power of 2") elif N