

Digital sound and discrete Fourier analysis

Øyvind Ryan

Jan 16, 2017

Euclidean inner product, Definition 2.1

For complex vectors of length N the Euclidean inner product is given by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=0}^{N-1} x_k \overline{y_k}.$$

The associated norm is

$$\|\mathbf{x}\| = \sqrt{\sum_{k=0}^{N-1} |x_k|^2}.$$

Discrete Fourier analysis, Definition 2.2

In Discrete Fourier analysis, a vector $\mathbf{x} = (x_0, \dots, x_{N-1})$ is represented as a linear combination of the N vectors

$$\phi_n = \frac{1}{\sqrt{N}} \left(1, e^{2\pi i n/N}, e^{2\pi i 2n/N}, \dots, e^{2\pi i kn/N}, \dots, e^{2\pi i n(N-1)/N} \right).$$

These vectors are called the normalised complex exponentials, or the pure digital tones of order N . n is also called frequency index. The whole collection $\mathcal{F}_N = \{\phi_n\}_{n=0}^{N-1}$ is called the N -point Fourier basis.

the N -point Fourier basis is an orthonormal basis for \mathbb{R}^N .

We will denote the change of coordinates matrix from the standard basis of \mathbb{R}^N to the Fourier basis \mathcal{F}_N by F_N . We will also call this the (N -point) *Fourier matrix*.

The matrix $\sqrt{N}F_N$ is also called the (N -point) *discrete Fourier transform*, or DFT. If \mathbf{x} is a vector in \mathbb{R}^N , then $\mathbf{y} = \text{DFT}\mathbf{x}$ are called the DFT coefficients of \mathbf{x} . (the DFT coefficients are thus the coordinates in \mathcal{F}_N , scaled with \sqrt{N}). $\text{DFT}\mathbf{x}$ is sometimes written as $\hat{\mathbf{x}}$.

The Fourier matrix is unitary

Theorem ??: The Fourier matrix F_N is the unitary $N \times N$ -matrix with entries given by

$$(F_N)_{nk} = \frac{1}{\sqrt{N}} e^{-2\pi i nk/N},$$

for $0 \leq n, k \leq N - 1$.

Definition ??: The matrix $\overline{F_N}/\sqrt{N}$ is the inverse of the matrix $\text{DFT} = \sqrt{N}F_N$. We call this inverse matrix the *inverse discrete Fourier transform*, or IDFT.

Direct implementation of the DFT in Matlab

```
function y = DFTImpl(x)
    N = size(x, 1);
    y = zeros(size(x));
    for n = 1:N
        D = exp(*2*pi*1i*(n-1)*(0:(N-1))/N);
        y(n) = dot(D, x);
    end
```

n has been replaced by $n - 1$ in this code since n runs from 1 to N (array indices must start at 1 in Matlab).

Direct implementation of the DFT in Python

```
def DFTImpl(x):  
    y = zeros_like(x).astype(complex)  
    N = len(x)  
    for n in xrange(N):  
        D = exp(-2*pi*n*1j*arange(float(N))/N)  
        y[n] = dot(D, x)  
    return y
```

Properties of the DFT, Theorem ??

Let \mathbf{x} be a real vector of length N . The DFT has the following properties:

- 1 $(\hat{\mathbf{x}})_{N-n} = \overline{(\hat{\mathbf{x}})_n}$ for $0 \leq n \leq N-1$.
- 2 If $x_k = x_{N-k}$ for all n (so \mathbf{x} is symmetric), then $\hat{\mathbf{x}}$ is a real vector.
- 3 If $x_k = -x_{N-k}$ for all k (so \mathbf{x} is antisymmetric), then $\hat{\mathbf{x}}$ is a purely imaginary vector.
- 4 If d is an integer and \mathbf{z} is the vector with components $z_k = x_{k-d}$ (the vector \mathbf{x} with its elements delayed by d), then $(\hat{\mathbf{z}})_n = e^{-2\pi i d n / N} (\hat{\mathbf{x}})_n$.
- 5 If d is an integer and \mathbf{z} is the vector with components $z_k = e^{2\pi i d k / N} x_k$, then $(\hat{\mathbf{z}})_n = (\hat{\mathbf{x}})_{n-d}$.

Relation between Fourier coefficients and DFT coefficients,

Proposition 2.9

Let $N > 2M$, $f \in V_{M,T}$, and let $\mathbf{x} = \{f(kT/N)\}_{k=0}^{N-1}$ be N uniform samples from f over $[0, T]$. The Fourier coefficients z_n of f can be computed from

$$(z_0, z_1, \dots, z_M, \underbrace{0, \dots, 0}_{N-(2M+1)}, z_{-M}, z_{-M+1}, \dots, z_{-1}) = \frac{1}{N} \text{DFT}_N \mathbf{x}.$$

In particular, the total contribution in f from frequency n/T , for $0 \leq n \leq M$, is given by y_n and y_{N-n} , where \mathbf{y} is the DFT of \mathbf{x} .

Proposition 2.12: Any $f \in V_{M,T}$ can be reconstructed uniquely from a uniform set of samples $\{f(kT/N)\}_{k=0}^{N-1}$, as long as $f_s > 2|\nu|$, where ν denotes the highest frequency in f .

Sampling theorem and the ideal interpolation formula for periodic functions, Theorem 2.13

Let f be a periodic function with period T , and assume that f has no frequencies higher than ν Hz. Then f can be reconstructed exactly from its samples $f(-MT_s), \dots, f(MT_s)$ (where T_s is the sampling period, $N = \frac{T}{T_s}$ is the number of samples per period, and $M = 2N + 1$) when the sampling rate $f_s = \frac{1}{T_s}$ is bigger than 2ν . Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-M}^M f(kT_s) \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)}.$$

Sampling theorem and the ideal interpolation formula, general version, Theorem 2.14

Assume that f has no frequencies higher than ν Hz. Then f can be reconstructed exactly from its samples $\dots, f(-2T_s), f(-T_s), f(0), f(T_s), f(2T_s), \dots$ when the sampling rate is bigger than 2ν . Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-\infty}^{\infty} f(kT_s) \frac{\sin(\pi(t - kT_s)/T_s)}{\pi(t - kT_s)/T_s}.$$

Using the DFT to adjust frequencies in sound, Example 2.16

```
[x, fs] = forw_comp_rev_DFT(13000, 1);  
playerobj=audioplayer(x, fs);  
playblocking(playerobj);
```

Compression by zeroing out DFT coefficients, Example 2.17

```
[x, fs] = forw_comp_rev_DFT(threshold=20);  
playerobj=audioplayer(x, fs);  
playblocking(playerobj);
```

```
[x, fs] = forw_comp_rev_DFT(n=3);  
playerobj=audioplayer(x, fs);  
playblocking(playerobj);
```

FFT algorithm when N is even Theorem 2.15

Let $\mathbf{y} = \text{DFT}_N \mathbf{x}$ be the N -point DFT of \mathbf{x} , with N an even number, and let $D_{N/2}$ be the $(N/2) \times (N/2)$ -diagonal matrix with entries $(D_{N/2})_{n,n} = e^{-2\pi i n / N}$ for $0 \leq n < N/2$. Then we have that

$$\begin{aligned}(y_0, y_1, \dots, y_{N/2-1}) &= \text{DFT}_{N/2} \mathbf{x}^{(e)} + D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)} \\ (y_{N/2}, y_{N/2+1}, \dots, y_{N-1}) &= \text{DFT}_{N/2} \mathbf{x}^{(e)} - D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)}\end{aligned}$$

where $\mathbf{x}^{(e)}, \mathbf{x}^{(o)} \in \mathbb{R}^{N/2}$ consist of the even- and odd-indexed entries of \mathbf{x} , respectively, i.e.

$$\mathbf{x}^{(e)} = (x_0, x_2, \dots, x_{N-2}) \quad \mathbf{x}^{(o)} = (x_1, x_3, \dots, x_{N-1}).$$

Let N be an even number and let $\tilde{\mathbf{x}} = \overline{\text{DFT}_N \mathbf{y}}$. Then we have that

$$\begin{aligned}(\tilde{x}_0, \tilde{x}_1, \dots, \tilde{x}_{N/2-1}) &= \overline{\text{DFT}_{N/2} \mathbf{y}^{(e)}} + \overline{D_{N/2} \text{DFT}_{N/2}} \mathbf{y}^{(o)} \\(\tilde{x}_{N/2}, \tilde{x}_{N/2+1}, \dots, \tilde{x}_{N-1}) &= \overline{\text{DFT}_{N/2} \mathbf{y}^{(e)}} - \overline{D_{N/2} \text{DFT}_{N/2}} \mathbf{y}^{(o)}\end{aligned}$$

where $\mathbf{y}^{(e)}, \mathbf{y}^{(o)} \in \mathbb{R}^{N/2}$ are the vectors

$$\mathbf{y}^{(e)} = (y_0, y_2, \dots, y_{N-2}) \quad \mathbf{y}^{(o)} = (y_1, y_3, \dots, y_{N-1}).$$

Moreover, $\mathbf{x} = \text{IDFT}_N \mathbf{y}$ can be computed from

$$\mathbf{x} = \tilde{\mathbf{x}}/N = \overline{\text{DFT}_N \mathbf{y}}/N$$

We have that

$$\begin{aligned}\text{DFT}_N \mathbf{x} &= \begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/2} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(o)} \end{pmatrix} \\ \text{IDFT}_N \mathbf{y} &= \frac{1}{N} \overline{\begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix}} \begin{pmatrix} \overline{\text{DFT}_{N/2}} & \mathbf{0} \\ \mathbf{0} & \overline{\text{DFT}_{N/2}} \end{pmatrix} \begin{pmatrix} \mathbf{y}^{(e)} \\ \mathbf{y}^{(o)} \end{pmatrix}\end{aligned}$$

Iterating the factorization 1

$$\text{DFT}_N \mathbf{x} = \begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix} \begin{pmatrix} I & D_{N/4} & 0 & 0 \\ I & -D_{N/4} & 0 & 0 \\ 0 & 0 & I & D_{N/4} \\ 0 & 0 & I & -D_{N/4} \end{pmatrix} \times$$

$$\begin{pmatrix} \text{DFT}_{N/4} & 0 & 0 & 0 \\ 0 & \text{DFT}_{N/4} & 0 & 0 \\ 0 & 0 & \text{DFT}_{N/4} & 0 \\ 0 & 0 & 0 & \text{DFT}_{N/4} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(ee)} \\ \mathbf{x}^{(eo)} \\ \mathbf{x}^{(oe)} \\ \mathbf{x}^{(oo)} \end{pmatrix}$$

where the vectors $\mathbf{x}^{(e)}$ and $\mathbf{x}^{(o)}$ have been further split into even- and odd-indexed entries. Clearly, if this factorization is repeated, we obtain a factorization

Iterating the factorization 2

$$\text{DFT}_N = \prod_{k=1}^{\log_2 N} \begin{pmatrix} I & D_{N/2^k} & 0 & 0 & \cdots & 0 & 0 \\ I & -D_{N/2^k} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & I & D_{N/2^k} & \cdots & 0 & 0 \\ 0 & 0 & I & -D_{N/2^k} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & I & D_{N/2^k} \\ 0 & 0 & 0 & 0 & \cdots & I & -D_{N/2^k} \end{pmatrix} P. \quad (1)$$

FFT implementation

```
function y = FFTImpl(x, FFTKernel)
    x = bitreverse(x);
    y = FFTKernel(x);
```

```
function y = FFTKernelStandard(x)
    N = size(x, 1);
    if N == 1
        y = x;
    else
        xe = FFTKernelStandard(x(1:(N/2)));
        xo = FFTKernelStandard(x((N/2+1):N));
        D = exp(-2*pi*1j*(0:(N/2-1))'/N);
        xo = xo.*D;
        y = [ xe + xo; xe - xo];
    end
```

```
y = FFTImpl(x, @FFTKernelStandard);
```