Chapter 7

Fourier transform

7.1 Recap of Fourier analysis

This section recalls material from the Analysis IV course.

Definition 7.1 *Let $L > 0$. A function $f : \mathbb{R} \to \mathbb{R}$ or $f : \mathbb{R} \to \mathbb{C}$ is called* L-periodic *if*

$$f(x + L) = f(x) \qquad \forall\ x \in \mathbb{R}.$$

In principle, any function on a bounded interval, $f : [a, b) \to \mathbb{R}$, could be extended to an L-periodic function with $L = b - a$ by setting $f(x + kL) = f(x)$ for $x \in [a, b)$ and $k \in \mathbb{Z}$. However, this is not a very useful way to think about periodic functions, because such extensions will rarely turn out to be continuous, differentiable, ... on $\mathbb{R}$.

Clearly, the functions $\sin(x)$, $\cos(x)$, $e^{\mathrm{i}x}$ are 2π-periodic. In the following, we will assume that $L = 2\pi$ without loss of generality (after a suitable rescaling of x).

For a 2π-periodic function $f : \mathbb{R} \to \mathbb{R}$ we can identify f with its restriction on any interval of length 2π. Let us take, for instance, $[0, 2\pi]$. *In the following, we will simply say "periodic function $f : [0, 2\pi] \to \mathbb{R}$".*

From the Analysis IV course, it is known that the Fourier series

$$\sum_{k=-\infty}^{+\infty} c_k e^{\mathrm{i}kx}, \quad c_k = \frac{1}{2\pi}\int_0^{2\pi} f(x) e^{-\mathrm{i}kx}\,\mathrm{d}x, \qquad \forall\ k \in \mathbb{Z}. \tag{7.1}$$

converges to f under certain conditions in a certain sense. For example, if $f \in L^2(0, 2\pi)$ then the truncated functions

$$f_N(x) := \sum_{k=-N}^{N} c_k e^{\mathrm{i}kx} \tag{7.2}$$

converge to f in the L^2-norm.

Lemma 7.2 (Orthogonality relations) *For every $k, \ell \in \mathbb{Z}$,*

$$\langle e^{\mathrm{i}\ell\cdot}, e^{\mathrm{i}k\cdot}\rangle_{L^2} := \int_0^{2\pi} e^{-\mathrm{i}\ell x} e^{\mathrm{i}kx}\,\mathrm{d}x = \begin{cases} 0 & \text{if } k \neq \ell, \\ 2\pi & \text{if } k = \ell. \end{cases}$$

Proof. See Analysis IV. □

Lemma 7.2 implies that $\left(e^{\mathrm{i}kx}\right)_{k\in\mathbb{Z}}$ is an orthogonal basis of $L^2(0,2\pi)$. By the Plancherel / Parseval theorem[20], we obtain for $f \in L^2(0,2\pi)$ that

$$\|f\|_{L_2}^2 = 2\pi \sum_{k=-\infty}^{\infty} |c_k|^2,$$

with the Fourier coefficients c_k defined as in (7.1). As a consequence, approximating f with the truncated Fourier series f_N in (7.9) results in the approximation error

$$\|f - f_N\|_{L_2}^2 = 2\pi \sum_{|k|>N} |c_k|^2. \tag{7.3}$$

Note that f_N can be represented by the $2N+1$ complex numbers $c_{-N}, \ldots, c_N$, where $c_{-k} = \overline{c_k}$ (and $c_0 \in \mathbb{R}$). It is worth noting that f_N is still real; using Euler's formula one has for $x \in \mathbb{R}$ that

$$\begin{aligned} f_N(x) &= \sum_{k=-N}^{N} c_k e^{\mathrm{i}kx} = \sum_{k=-N}^{N} c_k(\cos(kx) + \mathrm{i}\sin(kx)) \\ &= c_0 + \sum_{k=1}^{N} (c_k + \overline{c_k})\cos(kx) + \mathrm{i}(c_k - \overline{c_k})\sin(kx) \\ &= \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(kx) + b_k \sin(kx), \end{aligned} \tag{7.4}$$

where $a_0 = 2c_0$, $a_k = 2 \cdot \mathsf{Re}(c_k)$, $b_k = -2 \cdot \mathsf{Im}(c_k)$.

Let us recall the definition of the most famous integral transform.

Definition 7.3 *Given $f \in L^1(0,2\pi)$, we define the* Fourier transform *of f as*

$$\hat{f}(\xi) := \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-\mathrm{i}\xi x}\,\mathrm{d}x \qquad \forall\ x \in \mathbb{R}. \tag{7.5}$$

Comparing (7.1) and (7.5), we observe that $c_k = \hat{f}(k)$ for every $k \in \mathbb{Z}$.

7.2 Regularity and Fourier coefficients*

As we have seen in (7.3), the size of the Fourier coefficients for $k \to \infty$ determines how well f can be approximated by truncated Fourier series. From the *Riemann-Lebesgue Lemma* we know that $\hat{f}(k) \to 0$ as $|k| \to +\infty$ for $f \in L^1(0,2\pi)$. If

[20] In Analysis IV, this theorem might have been presented for the period $L = 1$, for which the pre-factor 2π is not needed.

extra regularity on f is assumed then it is possible to say more about the speed of convergence of its Fourier coefficients.

Proposition 7.4 *Let f be 2π-periodic and m times continuously differentiable on $\mathbb{R}$. Then*

$$|\hat{f}(k)| \le C_m |k|^{-m-1} \qquad \forall\, k \in \mathbb{Z},$$

where $C_m := \frac{1}{2\pi}\int_0^{2\pi} |f^{(m+1)}(x)|\,\mathrm{d}x$.

Proof. Given $k \in \mathbb{Z}$, one computes

$$\hat{f}(k) = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-\mathrm{i}kx}\,\mathrm{d}x = \frac{1}{2\pi \mathrm{i}k}\int_0^{2\pi} f'(x)e^{-\mathrm{i}kx}\,\mathrm{d}x = \frac{1}{\mathrm{i}k}\widehat{f'}(k),$$

where we used that the boundary term from the integration by parts vanishes since f and e^{-ikx} are 2π-periodic. By reiterating this procedure, it follows that

$$\hat{f}(k) = \frac{1}{(\mathrm{i}k)^{m+1}}\widehat{f^{(m+1)}}(k),$$

Because $f^{(m+1)}$ is continuous it is also in L^1. In turn, we can estimate

$$|\hat{f}(k)| \le \frac{1}{2\pi|k|^{m+1}}\int_0^{2\pi} |f^{(m+1)}(x)|\,\mathrm{d}x,$$

which completes the proof. □

If f is infinitely often differentiable then it follows from Proposition 7.4 that $c_k = \hat{f}(k)$ decays faster than $|k|^{-m}$ for *any* $m \in \mathbb{N}$. This superpolynomial convergence is improved further under the stronger assumption that f is real analytic. In this case, using tools from complex/harmonic analysis, it can be shown that there exist constants $\rho > 1$ and $C > 0$ such that

$$|\hat{f}(k)| \le C\rho^{-|k|} \qquad \forall\, k \in \mathbb{Z}. \tag{7.6}$$

7.3 The discrete Fourier transform (DFT)

The discrete Fourier transform (DFT) is a discrete analogue of the formula (7.5), that is, the transformation of a vector instead of a function.

Let us assume that the function f is sampled uniformly, that is, f is only known at the following set of n points in $[0, 2\pi]$:

$$x_j = \frac{2\pi j}{n}, \qquad y_j = f(x_j), \qquad \forall\, j = 0, 1, \ldots, n-1.$$

In analogy to (7.5), the *DFT* transforms these n function values into

$$z_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \qquad \forall\, k = 0, \ldots, n-1, \tag{7.7}$$

where $\omega_n := e^{-\frac{2\pi \mathrm{i}}{n}}$ is an nth root of unity. Up to scaling, the DFT can be obtained by applying the composite trapezoidal rule to approximate (7.5) using the integration nodes $(x_j)_{j=0}^{n-1}$ (EFY).

Let us recall that $\big(\omega_n^{-k}\big)_{k=0}^{n-1}$ forms an Abelian group, the so called group of nth roots of unity. Another important property is the discrete analogue of the orthogonality relations shown in Lemma 7.2.

Lemma 7.5 (Discrete orthogonality relations) *For every $k, \ell = 0, 1, \dots, n-1$, it holds that*

$$\sum_{j=0}^{n-1} \omega_n^{kj}\omega_n^{-\ell j} = \begin{cases} 0 & \text{if} \quad k \neq \ell, \\ n & \text{if} \quad k = \ell. \end{cases}$$

Proof. EFY. □

By defining the vectors $\mathbf{z} = \begin{pmatrix} z_0 & z_1 & \cdots & z_{n-1}\end{pmatrix}^\top$ and $\mathbf{y} = \begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1}\end{pmatrix}^\top$, the DFT (7.7) can be expressed as the matrix-vector product

$$\mathbf{z} = F_n \mathbf{y},$$

with the so called *Fourier matrix*

$$F_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} = (f_{kj})_{j,k=0}^{n-1}. \tag{7.8}$$

Note that it is custom to count vector/matrix indices from zero in the context of the DFT.

We now let a_{kl} denote the entry (j,k) (again counting from 0) of the matrix $F_n F_n^{\mathsf{H}}$ and obtain from Lemma 7.5 that

$$a_{k\ell} = \sum_{j=0}^{n-1} f_{kj}\overline{f_{\ell j}} = \sum_{i=0}^{n-1} \omega_n^{kj}\omega_n^{-\ell j} = \begin{cases} 0 & \text{if} \quad k \neq \ell, \\ n & \text{if} \quad k = \ell. \end{cases}$$

Hence, we get $F_n F_n^{\mathsf{H}} = nI_n$ or, in other words, $\frac{1}{\sqrt{n}}F_n$ is unitary! Therefore,

$$F_n^{-1} = \frac{1}{n}F_n^{\mathsf{H}} = \frac{1}{n}\overline{F}_n.$$

Given the transformed vector $\mathbf{z}$, this allows us to recover the data $\mathbf{y}$ from its DFT $\mathbf{z}$ using

$$\mathbf{y} = F_n^{-1}\mathbf{z} = \frac{1}{n}\overline{F}_n\mathbf{z}.$$

In terms of the entries, this Inverse Discrete Fourier Transform (IDFT) takes the form

$$y_k = \frac{1}{n}\sum_{j=0}^{n-1} z_j\omega_n^{-kj} \qquad \forall\ k = 0, \dots, n-1.$$

Example 7.6 (Sound digitization and data compression) *For carrying out this compression in* PYTHON, *we first need to load a file (called audio.mat in the following code snippet) containing an audio signal:*

```
from scipy.io import loadmat
from IPython.display import Audio
audio =  loadmat("audio.mat") # Load audio signal stored in audio.mat
y = audio['y']
Fs = audio['Fs'][0][0]
y = y[:,0]
```

To play this audio signal:

```
Audio(y,rate = Fs)
```

The commands

```
import scipy.fft as fft
z = fft.fft(y);
```

compute the DFT of the audio samples contained in the vector `y`. *From Figure 7.1, it is evident that the absolute values of the Fourier transform are small in large parts of the frequency range. We neglect these parts using*

```
ztilde = z * (abs(z)>30);
```

Using

```
ytilde = fft.ifft(ztilde)
Audio(ytilde,rate = Fs)
```

we compute the inverse transform and play the compressed signal. While the signal looks visually different, the sound does not seem to change too much.

One problem we neglected in the whole discussion above is that audio signals are usually not *periodic, which may lead to undesirable effects when blindly applying Fourier transforms. One way to fix this is to pad audio signals with zeros before performing the transform.* ◇

7.4 Resolving a mystery about the composite trapezoidal rule*

For a 2π-periodic function f we consider the approximation of the integral

$$\int_0^{2\pi} f(x)\,\mathrm{d}x.$$

Because of periodicity, the composite trapezoidal rule takes the form

$$Q_h^{(1)}[f] = \frac{2\pi}{N}\sum_{j=0}^{N-1} f(j/N)$$

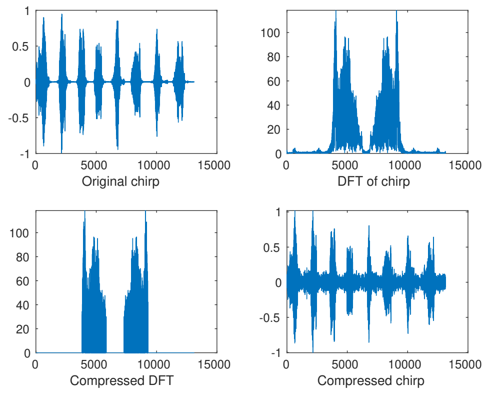


Figure 7.1: Compression of the chirp signal (from Matlab). Displayed is the original signal (top left), the absolute values of the DFT (top right), the compressed DFT (bottom left), and the compressed signal obtained from the IDFT of the compressed DFT (bottom right).

with $h = 2\pi/N$. Let us now consider the truncated Fourier expansion

$$f_{N-1}(x) := \sum_{k=-N+1}^{N-1} c_k e^{\mathrm{i}kx}. \tag{7.9}$$

On the one hand, we have

$$\int_0^{2\pi} e^{\mathrm{i}kx}\,\mathrm{d}x = \begin{cases} 0 & \text{for } k \neq 0, \\ 2\pi & \text{for } k = 0. \end{cases}$$

On the other hand, Lemma 7.5 yields

$$Q_h^{(1)}[e^{\mathrm{i}k\cdot}] = \frac{2\pi}{N}\sum_{j=0}^{N-1} e^{2\pi \mathrm{i}jk/N} = \begin{cases} 0 & \text{for } k = -N+1,\dots,-1,1,\dots,N-1, \\ 2\pi & \text{for } k = 0. \end{cases}$$

Hence, the composite trapezoidal rule with $h = 2\pi/N$ integrates the truncated Fourier expansion f_N *exactly*! This yields the following error bound:

$$\begin{aligned} & \left| \int_0^{2\pi} f(x)\,\mathrm{d}x - Q_h^{(1)}[f] \right| \\ \leq\ & \left| \int_0^{2\pi} (f(x) - f_N(x))\,\mathrm{d}x - Q_h^{(1)}[f - f_N] \right| \\ \leq\ & 4\pi \sum_{|k|>N} |c_k|. \end{aligned}$$

For a real analytic 2π-periodic function, the result (7.6) shows that $|c_k|$ decays exponentially fast and, in turn, the error of the composite trapezoidal rule also converges exponentially fast to zero.

7.5 The fast Fourier transform (FFT)

In the following, we will describe a fast algorithm for performing the DFT (and, at the same time, the IDFT). Recall that $\omega_n = e^{-2\pi \mathrm{i}/n}$. Then

$$\omega_n^k = \omega_n^{k+n} \ \forall k \in \mathbb{Z}, \quad \omega_n^n = 1, \quad \omega_n^{n/2} = -1, \tag{7.10}$$

Recall that the DFT is performed by multiplying a vector with the matrix $F_n = \big(\omega_n^{jk}\big)_{j,k=0}^{n-1}$. This matrix has a lot of structure, which can be exploited to accelerate matrix-vector multiplication. We illustrate this structure for $n = 6$. Then

$$F_6 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6 & \omega_6^2 & \omega_6^3 & \omega_6^4 & \omega_6^5 \\ 1 & \omega_6^2 & \omega_6^4 & \omega_6^6 & \omega_6^8 & \omega_6^{10} \\ 1 & \omega_6^3 & \omega_6^6 & \omega_6^9 & \omega_6^{12} & \omega_6^{15} \\ 1 & \omega_6^4 & \omega_6^8 & \omega_6^{12} & \omega_6^{16} & \omega_6^{20} \\ 1 & \omega_6^5 & \omega_6^{10} & \omega_6^{15} & \omega_6^{20} & \omega_6^{25} \end{pmatrix},$$

where $\omega_6 = e^{-2\pi \mathrm{i}/6} = e^{\pi \mathrm{i}/3}$. We reorder the rows such that the rows with even index appear first and then the rows with odd index. This can be achieved by defining

$$P_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

and applying its transpose to F_6:

$$P_6^T F_6 = \left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6^2 & \omega_6^4 & \omega_6^6 & \omega_6^8 & \omega_6^{10} \\ 1 & \omega_6^4 & \omega_6^8 & \omega_6^{12} & \omega_6^{16} & \omega_6^{20} \\ \hline 1 & \omega_6 & \omega_6^2 & \omega_6^3 & \omega_6^4 & \omega_6^5 \\ 1 & \omega_6^3 & \omega_6^6 & \omega_6^9 & \omega_6^{12} & \omega_6^{15} \\ 1 & \omega_6^5 & \omega_6^{10} & \omega_6^{15} & \omega_6^{20} & \omega_6^{25} \end{array}\right) = \left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6^2 & \omega_6^4 & 1 & \omega_6^2 & \omega_6^4 \\ 1 & \omega_6^4 & \omega_6^2 & 1 & \omega_6^4 & \omega_6^2 \\ \hline 1 & \omega_6 & \omega_6^2 & -1 & -\omega_6 & -\omega_6^2 \\ 1 & \omega_6^3 & \omega_6^6 & -1 & -\omega_6^3 & -\omega_6^6 \\ 1 & \omega_6^5 & \omega_6^4 & -1 & -\omega_6^5 & -\omega_6^4 \end{array}\right),$$

where we used the relation (7.10) multiple times. Because of $\omega_3^k = \omega_6^{2k}$ for $k \in \mathbb{Z}$, it follows that

$$P_6^T F_6 = \left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_3^1 & \omega_3^2 & 1 & \omega_3^1 & \omega_3^2 \\ 1 & \omega_3^2 & \omega_3^1 & 1 & \omega_3^2 & \omega_3^1 \\ \hline 1 & \omega_6 & \omega_6^2 & -1 & -\omega_6 & -\omega_6^2 \\ 1 & \omega_6\omega_3^1 & \omega_6^2\omega_3^2 & -1 & -\omega_6\omega_3^1 & -\omega_6^2\omega_3^2 \\ 1 & \omega_6\omega_3^2 & \omega_6^2\omega_3^1 & -1 & -\omega_6\omega_3^2 & -\omega_6^2\omega_3^1 \end{array}\right) = \begin{pmatrix} F_3 & F_3 \\ F_3\Omega_3 & -F_3\Omega_3 \end{pmatrix}$$

with

$$F_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega_3^1 & \omega_3^2 \\ 1 & \omega_3^2 & \omega_3^1 \end{pmatrix}, \quad \Omega_3 = \begin{pmatrix} 1 & & \\ & \omega_6 & \\ & & \omega_6^2 \end{pmatrix}.$$

This shows, for $n = 6$, that F_n is composed of four blocks $F_{n/2}$, combined with diagonal scaling and permutation. This relation generalizes to arbitrary even n.

Theorem 7.7 *Let $n \geq 2$ be even. Let P_n be the permutation matrix belonging to the permutation $\xi : \{0, \dots, n-1\} \to \{0, \dots, n-1\}$ with*

$$\xi : 0 \mapsto 0,\ 1 \mapsto 2,\ \dots,\ \frac{n}{2} - 1 \mapsto n-2,\ \frac{n}{2} \mapsto 1,\ \frac{n}{2} + 1 \mapsto 3,\ \dots,\ n-1 \mapsto n-1.$$

Then

$$P_n^{\mathsf{T}} F_n = \begin{pmatrix} F_{n/2} & F_{n/2} \\ F_{n/2}\Omega_{n/2} & -F_{n/2}\Omega_{n/2} \end{pmatrix} = \begin{pmatrix} F_{n/2} & \\ & F_{n/2} \end{pmatrix} \begin{pmatrix} I_{n/2} & I_{n/2} \\ \Omega_{n/2} & -\Omega_{n/2} \end{pmatrix}$$

with

$$\Omega_{n/2} = \operatorname{diag}(\omega_n^0, \omega_n^1, \dots, \omega_n^{n/2-1}).$$

Proof. This follows from a straightforward extension of the discussion above for $n = 6$, using relation (7.10). ☐

Theorem 7.7 can be used to perform a matrix-vector multiplication $F_n\mathbf{y}$ recursively. For this purpose, we partition the vector $\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$ such that $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{C}^{n/2}$. According to Theorem 7.7 we can write $F_n\mathbf{y}$ as follows:

$$F_n\mathbf{y} = P_n^T \begin{pmatrix} F_{n/2} & \\ & F_{n/2} \end{pmatrix} \begin{pmatrix} I_{n/2} & I_{n/2} \\ \Omega_{n/2} & -\Omega_{n/2} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = P_n^T \begin{pmatrix} F_{n/2}(\mathbf{y}_1 + \mathbf{y}_2) \\ F_{n/2}\Omega_{n/2}(\mathbf{y}_1 - \mathbf{y}_2) \end{pmatrix}.$$

In other words, the multiplication with F_n can be reduced to two multiplications with $F_{n/2}$ and some cheap additional computations. Applying this recursion repeatedly, one arrives at the following algorithm when n is a power of two.[21]

Algorithm 7.8

```
Input:  Vector $\mathbf{y} \in \mathbb{C}^n$ where $n$ is a power of 2.
Output: Matrix-vector product $\mathbf{z} = F_n\mathbf{y}$.

  Partition $\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$ with $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{C}^{n/2}$.
  $\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{y}_2$, $\mathbf{z}_2 = \Omega_{n/2}(\mathbf{y}_1 - \mathbf{y}_2)$.
  Recursion: $\mathbf{z}_1 \leftarrow F_{n/2}\mathbf{z}_1$
  Recursion: $\mathbf{z}_2 \leftarrow F_{n/2}\mathbf{z}_2$.
  $\mathbf{z} = P_n^T \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{pmatrix}$
```

Python

```python
import numpy as np
def myfft(y):
```

[21]It is instructive to verify with a small example how `reshape` is used to effect the multiplication with P_n.

```
    n = y.shape[0]
    if (n == 1):  return y
    omega = np.exp(-2*np.pi*1j/n);
    mid   = int(n/2)
    z1    = y[0:mid]+y[mid:n];
    z2    = np.power(omega, range(0,mid))*(y[0:mid]-y[mid:n])
    z     = np.concatenate((myfft(z1), myfft(z2)))
    z     = np.reshape(np.reshape(z, (2,mid)), (n,), 'F') # permute
    return z
```

It is worth reflecting how the permutation is realized efficiently in the Python code.

The computational complexity of recursive algorithms can often be derived using the Master theorem (see Wikipedia). In the case of Algorithm 7.8, it is also quite easy to estimate the complexity directly. Let $A(n)$ denote the number of operations required by Algorithm 7.8 for a vector of length n. Ignoring the cost for computing the powers of ω_n (which can be computed beforehand and stored in a table), then[22] $A(n) = 5n + 2\,A(n/2)$. Recursive application of this formula yields

$$A(n) = 5n + 5n + 4\,A(n/4) = \cdots = 5kn + 2^k A(n/2^k).$$

Setting $k = \log_2 n$ we have $A(n/2^k) = A(n/n) = A(1) = 0$ and hence

$$A(n) = 5n \log_2 n.$$

Because this compares very favorably with the $O(n^2)$ complexity of general matrix-vector multiplication, one calls Algorithm 7.8 the Fast Fourier Transform (FFT).

Remark 7.9 *In Python, both NumPy and SciPy have functions implementing the FFT; the one by SciPy tends to be faster. The software package FFTW (Fastest Fourier Transform in the West, see* `http://www.fftw.org/`*) is a popular and well tuned implementation of the FFT. For $n = 2^k$ the algorithm by FFTW roughly corresponds to Algorithm 7.8. For $n \neq 2^k$ one needs to resort to other decompositions/factorizations of n; the asymptotic complexity remains $O(n \log_2 n)$ but the constants can be significantly larger, especially if n has large prime factors. There is a Python wrapper for FFTW, called pyFFTW, which is more complicated to call. Once the package pyFFTW is installed, the FFT and inverse FFT can be compted as follows:*

```
import pyfftw
#assume y is our signal
target_len = len(y)
a = pyfftw.empty_aligned(target_len, dtype='complex128')
b = pyfftw.empty_aligned(target_len, dtype='complex128')
#build a fft object to be called later
fft_object = pyfftw.FFTW(a, b)
a[:] = y
```

[22]A complex addition or subtraction counts 2 flops and a complex multiplication counts 6 flops.

```
#call fft, both fft_a and b will be the result of fft
fft_a = fft_object()
c = pyfftw.empty_aligned(target_len, dtype='complex128')
#build an ifft object
ifft_object =  pyfftw.FFTW(b, c, direction='FFTW_BACKWARD')
#call ifft, both ifft_b and c will be the result of ifft
ifft_b = ifft_object()
```

7.6 Discrete cosine transform (DCT)

We now assume that f is not only 2π-periodic but also even, that is,

$$f(x+2k\pi) = f(x), \qquad f(x) = f(-x), \qquad \forall k \in \mathbb{Z}, x \in \mathbb{R}.$$

It is then sufficient to restrict f to the interval $[-\pi, \pi]$. Its Fourier series, if convergent, reads as

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(kx) \qquad \forall\, k \in \mathbb{Z}; \tag{7.11}$$

see (7.4). The following orthogonality relations hold:

Lemma 7.10 *For every $\ell, k \in \mathbb{Z}$*

$$\int_0^\pi \cos(\ell x)\cos(kx)\ \mathrm{d}x = \begin{cases} 0 & \text{if} \quad \ell \neq k, \\ \frac{\pi}{2} & \text{if} \quad \ell = k \neq 0, \\ \pi & \text{if} \quad \ell = k = 0. \end{cases}$$

The Fourier coefficients $(a_k)_{k\in\mathbb{Z}}$ can be recovered by multiplying (7.11) with $\cos(\ell x)$, integrating from 0 to π, and using Lemma 7.10, which yields

$$\int_0^\pi f(x)\cos(\ell x)\ \mathrm{d}x = \frac{a_0}{2}\int_0^\pi \cos(\ell x)\ \mathrm{d}x + \sum_{k=1}^{\infty} a_k \int_0^\pi \cos(kx)\cos(\ell x)\ \mathrm{d}x,$$

and thus

$$a_k = \frac{2}{\pi}\int_0^\pi f(x)\cos(kx)\ \mathrm{d}x \qquad \forall\, k \in \mathbb{N}. \tag{7.12}$$

Let us know suppose that f is known only at some discrete points in $[0, \pi]$:

$$x_j := \frac{(2j+1)\,\pi}{2N}, \qquad y_j := f(x_j) \qquad \forall\, j = 0, \ldots, N-1. \tag{7.13}$$

The discrete counterpart of (7.11) becomes

$$y_j = \frac{z_0}{2} + \sum_{k=1}^{N-1} z_k \cos(kx_j), \qquad \forall\, j = 0, \ldots, N-1, \tag{7.14}$$

and the following discrete orthogonality relations hold.

Lemma 7.11 *For all $\ell, k \in \mathbb{Z}$*

$$\sum_{j=0}^{N-1} \cos(\ell x_j)\cos(k x_j) = \begin{cases} 0 & \textit{if} \quad \ell \neq k, \\ \frac{N}{2} & \textit{if} \quad \ell = k \neq 0, \\ N & \textit{if} \quad \ell = k = 0. \end{cases}$$

Proof. EFY. □

We now aim at computing $(z_k)_{k=0}^{N-1}$ such that (7.14) holds. Mimicking the procedure above in a discrete setting, we multiply (7.14) with $\cos(\ell x_j)$, summing from $j = 0$ to $j = N-1$ and employ Lemma 7.11, which yields

$$\sum_{j=0}^{N-1} y_j \cos(\ell x_j) = \frac{z_0}{2} \sum_{j=0}^{N-1} \cos(\ell x_j) + \sum_{k=1}^{N-1} z_k \sum_{j=0}^{N-1} \cos(k x_j)\cos(\ell x_j).$$

Thus,

$$z_k = \frac{2}{N} \sum_{j=0}^{N-1} y_j \cos(k x_j) \qquad \forall\, k = 0, \dots, N-1. \tag{7.15}$$

Definition 7.12 (Discrete Cosine Transform) $\big(\hat{f}_N(k)\big)_{k\in\mathbb{Z}}$ *is called the* discrete cosine transform (DCT) *of f with respect to the discretization $(x_j)_{j=0}^{N-1}$ defined in* (7.13).

Remark 7.13 *It is possible to interpret the DCT as the approximation of* (7.12), *by using the composite midpoint quadrature rule.*

Definition 7.14 (Inverse Discrete Cosine Transform) *The sequence $(y_j)_{j\in\mathbb{Z}}$ is called* inverse discrete cosine transform (IDCT) *of f with respect to the discretization $(x_j)_{j=0}^{N-1}$.*

7.6.1 The JPEG: an image compression standard*

The light intensity measured by a camera is generally sampled over a rectangular array of picture elements called *pixels.* Let us consider an image consisting of M^2 pixels, such that each couple (i, j), for $i, j = 0, \dots, M-1$, corresponds to a pixel. For the sake of simplicity of the discussion, let us focus on the case of black and white pictures. A BW picture can be thought as a function $Y : M \times M \to \{0, \dots, 255\}$, $(i, j) \mapsto Y(i, j)$, where $Y(i, j)$ represents the gray level at the pixel (i, j). Thus, $M^2 \cdot 8$ bits (the number 255 is 11111111 in base 2) per pixel are needed in order to store a picture. In principle M may be very large: a typical high resolution color picture for the web contains on the order of one millions pixels. However, state-of-the-art techniques can compress typical images from 1/10 to 1/50 without visibly affecting image quality. One of the most popular procedures is indeed JPEG (N. Ahmed, T. Natarajan, K. R. Rao, 1974). Let us subdivide the image into 8×8 blocks. For each block $(Y_{i_k, j_\ell})_{k,\ell=0}^{7}$, where $(i_k)_{k=0}^{7}$, $(j_\ell)_{\ell=0}^{7} \subset \{0, \dots, M\}$ are subsequences of consecutive indices, we can apply the DCT, passing from the *spatial domain* to the *frequency domain.* In this way every 8×8 block of source image sample is effectively a discrete signal with 64 entries, which is a function of the two spatial dimensions, denoted for the sake of simplicity of the notation as $(Y_{i,j})_{i,j=0}^{N}$, with $N = 7$. By analogy to (7.14), we want to find $(Z_{k,\ell})_{k,\ell=0}^{N-1}$ such that

$$Y_{i,j} = \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} \tilde{Z}_{k,\ell} \cos(kx_i) \cos(x_j), \qquad i, j = 0, \ldots, N-1, \tag{7.16}$$

where, in order to compensate for the factor $1/2$, we employ the notation $\tilde{Z}_{0,0} = Z_{0,0}/4$, $\tilde{Z}_{k,0} = Z_{k,0}/2$, $\tilde{Z}_{0,\ell} = Z_{0,\ell}/2$, $\tilde{Z}_{k,\ell} = Z_{k,\ell}$, $k, \ell \geq 1$. In Figure 7.2 the reader can see the representation of the 64 basis functions $(\cos(kx_i)\cos(\ell x_j))_{k,\ell=0}^{N-1}$ on a single 8×8 block. In particular, the columns correspond to the index k and the rows to the index ℓ, $k, \ell = 0, 1, \ldots, N = 7$. Increasing k, respectively ℓ, corresponds to higher oscillations in the x-direction, respectively y-direction.

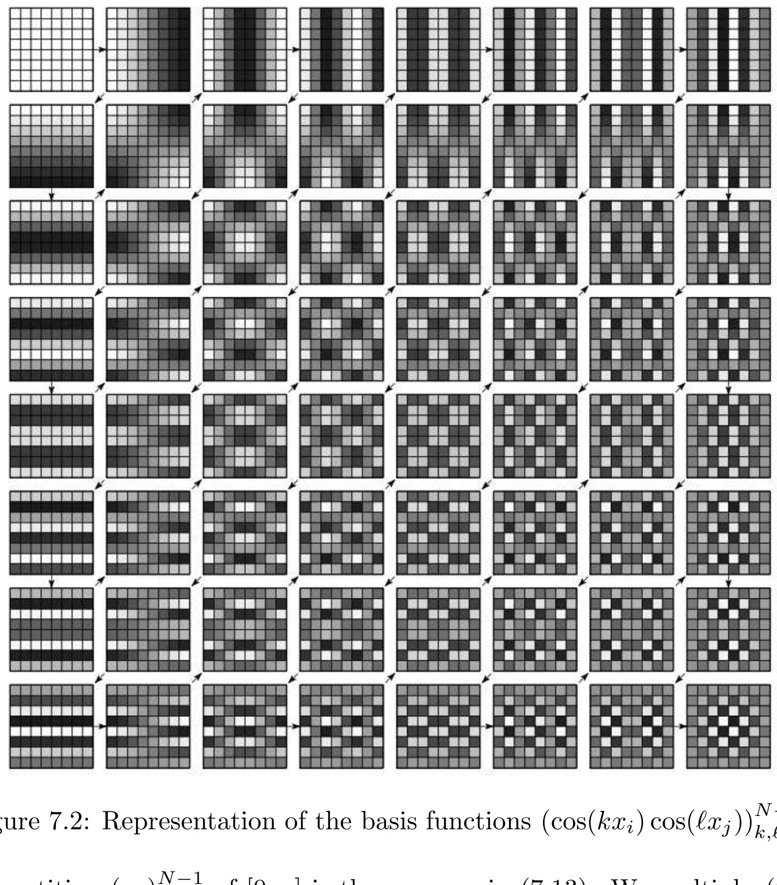


Figure 7.2: Representation of the basis functions $(\cos(kx_i)\cos(\ell x_j))_{k,\ell=0}^{N-1}$.

The partition $(x_j)_{j=0}^{N-1}$ of $[0, \pi]$ is the same as in (7.13). We multiply (7.16) by $\cos(kx_i)\cos(\ell x_j)$, sum over $i, j = 0, \ldots, N-1$ and use the discrete orthogonality relations (7.11). In this way we get the $2D$ counterpart of (7.15), that is

$$Z_{k,\ell} = \frac{4}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Y_{i,j} \cos(kx_i) \cos(\ell x_j), \qquad k, \ell = 0, \ldots, N-1. \tag{7.17}$$

The DCT takes the signal representing the block as an input and decomposes it into 64 orthogonal basis signals, each one of them corresponding to a particular *frequency*. The value of a frequency reflects the size and speed of a change as you can see from Figure 7.2. The output is the collection of 64 DCT coefficients, representing the *amplitudes* of these signals. The first coefficient, corresponding to the zero frequency in both spatial dimensions, is often called DC (*direct current*). The remaining 63 entries are called AC (*alternating currents*). The high frequencies represent the high contrast areas in the image, i.e. rapid changes in pixel intensity. Note that in a classic image there is a high continuity between pixel values. Hence it turns out that the numerically important AC coefficients can be found in the square 4×4 around the DC coefficient.

Once the DCT coefficients are obtained, we would like to numerically represent them with no greater precision than is necessary to achieve the desired image quality. This step is called *quantization* of the signal. Each of the 64 DCT coefficients is quantized according to a 8×8 matrix T called, *quantization matrix*, with integer entries between 1 and 255, which is specified by the user and is conceived to provide greater resolution to more perceptible frequency components on less perceptible ones. In formulas, the quantization step reads as

$$Z_{k,\ell} \mapsto \lfloor \frac{Z_{k,\ell}}{T_{k,\ell}} \rfloor, \qquad k, \ell = 0, \ldots, N-1. \tag{7.18}$$

Note that since the entries of T corresponding to the high frequencies are usually high and because we use the function “floor” in (7.18), the resulting high frequency coefficients will be zero. Let $\mathcal{Z}_N$ be the 8×8 matrix of coefficients after quantization. Let us go through its entries by following a *zig-zag path* (see Figure 7.3) in order to construct a vector of 64 coefficients. We just mention that this trick allows to further reduce the amount of information to be compressed of the image by placing low-frequency coefficients (more likely to be non-zero) before high frequency coefficients.

The procedure described above can be reversed by applying the IDCT which takes the encoded coefficients and reconstructs the image signal by summing the basis signals. However, because of the quantization step, there is an inevitable loss of information. We have indeed introduced a numerical error and we made the whole procedure irreversible. That is why the JPEG is said to be a *lossy compression* technique. In Figure 7.4a we can see an 8×8 block from an image and in Figure 7.4b its decoding process that follows the zig-zag path.

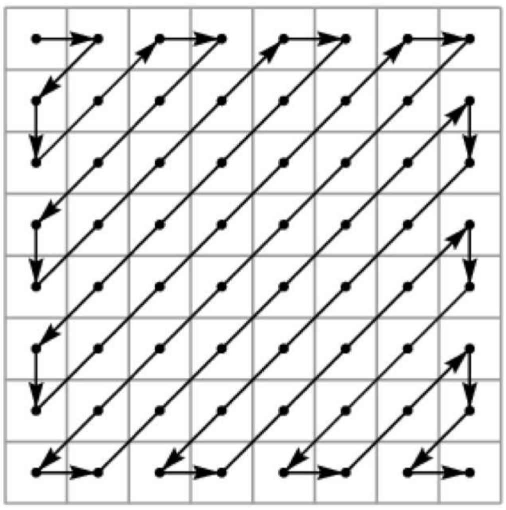
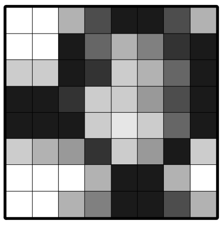
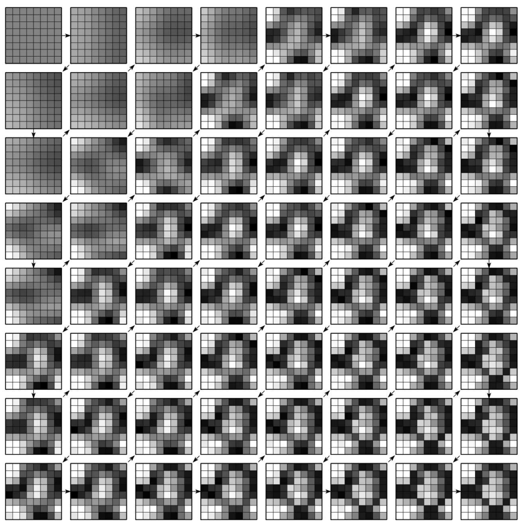


Figure 7.3: Zig-zag path used for the encoding of images in the JPEG.



(a) Target block.



(b) Decoding using the zig-zag procedure.

Figure 7.4: The reconstruction process of a sample 8 × 8 block from an image.