Chapter 4

Implementation of the DFT and the DCT

The main application of the DFT and the DCT is as tools to compute frequency information in large datasets. It is therefore important that these operations can be performed by efficient algorithms. Straightforward implementation from the definition is not efficient if the data sets are large. However, it turns out that the underlying matrices may be factored in a way that leads to much more efficient algorithms, and this is the topic of the present chapter.

4.1 The Fast Fourier Transform (FFT)

In this section we will discuss the most widely used implementation of the DFT, which is usually referred to as the Fast Fourier Transform (FFT). For simplicity, we will assume that N, the length of the vector that is to be transformed by the DFT, is a power of 2. In this case it is relatively easy to simplify the DFT algorithm via a factorisation of the Fourier matrix. The foundation is provided by a simple reordering of the DFT.

Theorem 4.1 (FFT algorithm)**.** Let $\boldsymbol{y} = F_N \boldsymbol{x}$ be the N-point DFT of $\boldsymbol{x}$ with N an even number. Foran any integer n in the interval $[0, N/2-1]$ the DFT $\boldsymbol{y}$ of $\boldsymbol{x}$ is then given by

$$y_n = \frac{1}{\sqrt{2}} \left((F_{N/2}\boldsymbol{x}^{(e)})_n + e^{-2\pi i n/N} (F_{N/2}\boldsymbol{x}^{(o)})_n \right), \tag{4.1}$$

$$y_{N/2+n} = \frac{1}{\sqrt{2}} \left((F_{N/2}\boldsymbol{x}^{(e)})_n - e^{-2\pi i n/N} (F_{N/2}\boldsymbol{x}^{(o)})_n \right), \tag{4.2}$$

where $\boldsymbol{x}^{(e)}, \boldsymbol{x}^{(o)}$ are the sequences of length $N/2$ consisting of the even and

odd samples of $\boldsymbol{x}$, respectively. In other words,

$$(\boldsymbol{x}^{(e)})_k = x_{2k} \text{ for } 0 \le k \le N/2-1,$$
$$(\boldsymbol{x}^{(o)})_k = x_{2k+1} \text{ for } 0 \le k \le N/2-1.$$

Put differently, the formulas (4.1)–(4.2) reduces the computation of an N-point DFT to 2 $N/2$-point DFT's. It turns out that this can speed up computations considerably, but let us first check that these formulas are correct.

Proof. Suppose first that $0 \le n \le N/2-1$. We start by splitting the sum in the expression for the DFT into even and odd indices,

$$\begin{aligned}
y_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n 2k/N} + \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n (2k+1)/N} \\
&= \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k/(N/2)} \\
&\qquad + e^{-2\pi i n/N} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k/(N/2)} \\
&= \frac{1}{\sqrt{2}} \left(F_{N/2} \boldsymbol{x}^{(e)} \right)_n + \frac{1}{\sqrt{2}} e^{-2\pi i n/N} \left(F_{N/2} \boldsymbol{x}^{(o)} \right)_n,
\end{aligned}$$

where we have substituted $\boldsymbol{x}^{(e)}$ and $\boldsymbol{x}^{(o)}$ as in the text of the theorem, and recognized the $N/2$-point DFT in two places. For the second half of the DFT coefficients, i.e. $\{y_{N/2+n}\}_{0 \le n \le N/2-1}$, we similarly have

$$\begin{aligned}
y_{N/2+n} &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i (N/2+n) k/N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-\pi i k} e^{-2\pi i n k/N} \\
&= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n 2k/N} - \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n (2k+1)/N} \\
&= \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k/(N/2)} \\
&\qquad - e^{-2\pi i n/N} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k/(N/2)} \\
&= \frac{1}{\sqrt{2}} \left(F_{N/2} \boldsymbol{x}^{(e)} \right)_n - \frac{1}{\sqrt{2}} e^{-2\pi i n/N} \left(F_{N/2} \boldsymbol{x}^{(o)} \right)_n.
\end{aligned}$$

This concludes the proof. □

It turns out that Theorem 4.1 can be interpreted as a matrix factorization. For this we need to define the concept of a block matrix.

Definition 4.2. Let $m_0, \ldots, m_{r-1}$ and $n_0, \ldots, n_{s-1}$ be integers, and let $A^{(i,j)}$ be an $m_i \times n_j$-matrix for $i = 0, \ldots, r-1$ and $j = 0, \ldots, s-1$. The notation

$$A = \left(\begin{array}{c|c|c|c} A^{(0,0)} & A^{(0,1)} & \cdots & A^{(0,s-1)} \\ \hline A^{(1,0)} & A^{(1,1)} & \cdots & A^{(1,s-1)} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline A^{(r-1,0)} & A^{(r-1,1)} & \cdots & A^{(r-1,s-1)} \end{array}\right)$$

denotes the $(m_0 + m_1 + \ldots + m_{r-1}) \times (n_0 + n_1 + \ldots + n_{s-1})$-matrix where the matrix entries occur as in the $A^{(i,j)}$ matrices, in the way they are ordered, and with solid lines indicating borders between the blocks. When A is written in this way it is referred to as a block matrix.

We will express the Fourier matrix in factored form involving block matrices. The following observation is just a formal way to split a vector into its even and odd components.

Observation 4.3. Define the permutation matrix P_N by

$$\begin{aligned} (P_N)_{i,2i} &= 1, \quad \text{for } 0 \le i \le N/2 - 1; \\ (P_N)_{i,2i-N+1} &= 1, \quad \text{for } N/2 \le i < N; \\ (P_N)_{i,j} &= 0, \quad \text{for all other } i \text{ and } j; \end{aligned}$$

and let $\boldsymbol{x}$ be a column vector. The mapping $\boldsymbol{x} \to P\boldsymbol{x}$ permutes the components of $\boldsymbol{x}$ so that the even components are placed first and the odd components last,

$$P_N \boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}^{(e)} \\ \boldsymbol{x}^{(o)} \end{pmatrix},$$

with $\boldsymbol{x}^{(e)}$, $\boldsymbol{x}^{(o)}$ defined as in Theorem 4.1.

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 $n = 0, 1, \ldots, N/2 - 1$. It is clear from Equation (4.1) that the first half of $\boldsymbol{y}$ is then given by obtained as

$$\frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & D_{N/2} F_{N/2} \end{array} \right) P_N \boldsymbol{x},$$

and from Equation (4.2) that the second half of $\boldsymbol{y}$ can be obtained as

$$\frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & -D_{N/2} F_{N/2} \end{array} \right) P_N \boldsymbol{x}.$$

From these two formulas we can derive the promised factorisation of the Fourier matrix.

Theorem 4.4 (DFT matrix factorization)**.** The Fourier matrix may be factored as

$$F_N = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} F_{N/2} & D_{N/2} F_{N/2} \\ \hline F_{N/2} & -D_{N/2} F_{N/2} \end{array} \right) P_N. \tag{4.3}$$

This factorization in terms of block matrices is commonly referred to as the *FFT factorization* of the Fourier matrix. In implementations, this factorization is typically repeated, so that $F_{N/2}$ is replaced with a factorization in terms of $F_{N/4}$, this again with a factorization in terms of $F_{N/8}$, and so on.

The input vector $\boldsymbol{x}$ to the FFT algorithm is mostly assumed to be real. In this case, the second half of the FFT factorization can be simplified, since we have shown that the second half of the Fourier coefficients can be obtained by symmetry from the first half. In addition we need the formula

$$y_{N/2} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N/2-1} \left((\boldsymbol{x}^{(e)})_n - (\boldsymbol{x}^{(o)})_n \right)$$

to obtain coefficient $\frac{N}{2}$, since this is the only coefficient which can't be obtained from $y_0, y_1, \ldots, y_{N/2-1}$ by symmetry.

In an implementation based on formula (4.3), we would first compute $P_N \boldsymbol{x}$, which corresponds to splitting $\boldsymbol{x}$ into the even-indexed and odd-indexed samples. The two leftmost blocks in the block matrix in (4.3) correspond to applying the $\frac{N}{2}$-point DFT to the even samples. The two rightmost blocks correspond to applying the $N/2$-point DFT to the odd samples, and multiplying the result with $D_{N/2}$. The results from these transforms are finally added together. By repeating the splitting we will eventually come to the case where $N = 1$. Then F_1 is just the scalar 1, so the DFT is the trivial assignment $y_0 = x_0$. The FFT can therefore be implemented by the following MATLAB code:

```
function y = FFTImpl(x)
N = length(x);
if N == 1
    y = x(1);
else
    xe = x(1:2:(N-1));
    xo = x(2:2:N);
    ye = FFTImpl(xe);
    yo = FFTImpl(xo);
    D=exp(-2*pi*1j*(0:N/2-1)'/N);
    y = [ ye + yo.*D; ye - yo.*D]/sqrt(2);
end
```

Note that this function is recursive; it calls itself. If this is you first encounter with a recursive program, it is worth running through the code for $N = 4$, say.

4.1.1 The Inverse Fast Fourier Transform (IFFT)

The IDFT is very similar to the DFT, and it is straightforward to prove the following analog to Theorem 4.1 and (4.3).

Theorem 4.5 (IDFT matrix factorization)**.** The inverse of the Fourier matrix can be factored as

$$(F_N)^H = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} (F_{N/2})^H & E_{N/2}(F_{N/2})^H \\ \hline (F_{N/2})^H & -E_{N/2}(F_{N/2})^H \end{array} \right) P_N, \tag{4.4}$$

where $E_{N/2}$ is the $(N/2) \times (N/2)$-diagonal matrix with entries given by $(E_{N/2})_{n,n} = e^{2\pi i n/N}$, for $n = 0, 1, \ldots, N/2 - 1$.

We note that the only difference between the factored forms of F_N and F_N^H is the positive exponent in $e^{2\pi i n/N}$. With this in mind it is straightforward to modify `FFTImpl.m` so that it performs the inverse DFT.

MATLAB has built-in functions for computing the DFT and the IDFT, called `fft` and `ifft`. Note, however, that these functions do not used the normalization $1/\sqrt{N}$ that we have adopted here. The MATLAB help pages give a short description of these algorithms. Note in particular that MATLAB makes no assumption about the length of the vector. MATLAB may however check if the length of the vector is 2^r, and in those cases a variant of the algorithm discussed here is used. In general, fast algorithms exist when the vector length N can be factored as a product of small integers.

Many audio and image formats make use of the FFT. To get optimal speed these algorithms typically split the signals into blocks of length 2^r with r some integer in the range 5–10 and utilise a suitable variant of the algorithms discussed above.

4.1.2 Reduction in the number of multiplications with the FFT

Before we continue we also need to explain why the FFT and IFFT factorizations lead to more efficient implementations than the direct DFT and IDFT implementations. We first need some terminology for how we count the number of operations of a given type in an algorithm. In particular we are interested in in the limiting behaviour when N becomes large, which is the motivation for the following definition.

Definition 4.6 (Order of an algorithm)**.** Let R_N be the number of operations of a given type (such as multiplication, addition) in an algorithm, where N describes the dimension of the data in the algorithm (such as the size of the matrix or length of the vector), and let f be a positive function. The algorithm is said to be of order N, which is written $O(f(N))$, if the number of operations

grows as $f(N)$ for large N, or more precisely, if

$$\lim_{N\to\infty} \frac{R_N}{f(N)} = c > 0.$$

We will also use this notation for functions, and say that a real function g is $O(f(\boldsymbol{x}))$ if $\lim g(\boldsymbol{x})/f(\boldsymbol{x}) = 0$ where the limit mostly will be taken as $\boldsymbol{x} \to \mathbf{0}$ (this means that $g(\boldsymbol{x})$ is much smaller than $f(\boldsymbol{x})$ when $\boldsymbol{x}$ approaches the limit).

Let us see how we can use this terminology to describe the complexity of the FFT algorithm. Let M_N be the number of multiplications needed by the N-point FFT as defined by Theorem 4.1. It is clear from the algorithm that

$$M_N = 2M_{N/2} + N/2. \tag{4.5}$$

The factor 2 corresponds to the two matrix multiplications, while the term $N/2$ denotes the multiplications in the exponent of the exponentials that make up the matrix $D_{N/2}$ (or $E_{N/2}$) — the factor $2\pi i/N$ may be computed once and for all outside the loops. We have not counted the multiplications with `1/sqrt(2)`. The reason is that, in most implementations, this factor is absorbed in the definition of the DFT itself.

Note that all multiplications performed by the FFT are complex. It is normal to count the number of real multiplications instead, since any multiplication of two complex numbers can be performed as four multiplications of real numbers (and two additions), by writing the number in terms of its real and imaginary part, and myltiplying them together. Therefore, if we instead define M_N to be the number of real multiplications required by the FFT, we obtain the alternative recurrence relation

$$M_N = 2M_{N/2} + 2N. \tag{4.6}$$

In Exercise 1 you will be asked to derive the solution of this equation and show that the number of real multiplications required by this algorithm is $O(2N \log_2 N)$. In contrast, the direct implementation of the DFT requires N^2 complex multiplications, and thus $4N^2$ real multiplications. The exact same numbers are found for the IFFT.

Theorem 4.7 (Number of operations in the FFT and IFFT algorithms)**.** The N-point FFT and IFFT algorithms both require $O(2N \log_2 N)$ real multiplications. In comparison, the number of real multiplications required by direct implementations of the N-point DFT and IDFT is $4N^2$.

In other words, the FFT and IFFT significantly reduce the number of multiplications, and one can show in a similar way that the number of additions required by the algorithm is also roughly $O(N \log_2 N)$. This partially explains the efficiency of the FFT algorithm. Another reason is that since the FFT splits the calculation of the DFT into computing two DFT's of half the size, the FFT

is well suited for parallel computing: the two smaller FFT's can be performed independently of one another, for instance in two different computing cores on the same computer.

Since filters are diagonalized by the DFT, it may be tempting to implement a filter by applying an FFT, multiplying with the frequency response, and then apply the IFFT. This is not usually done, however. The reason is that most filters have too few nonzero coefficients for this approach to be efficient — it is then better to use the direct algorithm for the DFT, since this may lead to fewer multiplications than the $O(N \log_2 N)$ required by the FFT.

Exercises for Section 4.1

Ex. 1 — In this exercise we will compute the number of real multiplications needed by the FFT algorithm given in the text. The starting point will be the difference equation (4.6) for the number of real multiplications for an N-point FFT.

a. Explain why $x_r = M_{2^r}$ is the solution to the difference equation $x_{r+1} - 2x_r = 4 \cdot 2^r$.

b. Show that the general solution to the difference equation is $x_r = 2r2^r + C2^r$.

c. Explain why $M_N = O(2N \log_2 N)$ (you do not need to write down the initial conditions for the difference equation in order to find the particular solution).

Ex. 2 — When we wrote down the difference equation $M_N = 2M_{N/2} + 2N$ for the number of multiplications in the FFT algorithm, you could argue that some multiplications were not counted. Which multiplications in the FFT algorithm were not counted when writng down this difference equation? Do you have a suggestion to why these multiplications were not counted?

Ex. 3 — Write down a difference equation for computing the number of real additions required by the FFT algorithm.

Ex. 4 — It is of course not always the case that the number of points in a DFT is $N = 2^n$. In this exercise we will see how we can attack the more general case.

a. Assume that N can be divided by 3, and consider the following splitting, which follows in the same way as the splitting used in the deduction of

the FFT-algorithm:

$$\begin{aligned} y_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i n k/N} \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k} e^{-2\pi i n 3k/N} + \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k+1} e^{-2\pi i n (3k+1)/N} \\ &+ \frac{1}{\sqrt{N}} \sum_{k=0}^{N/3-1} x_{3k+2} e^{-2\pi i n (3k+2)/N} \end{aligned}$$

Find a formula which computes $y_0, y_1, \ldots, y_{N/3-1}$ by performing 3 DFT's of size $N/3$.

b. Find similar formulas for computing $y_{N/3}, y_{N/3+1}, \ldots, y_{2N/3-1}$, and $y_{2N/3}, y_{2N/3}+1, \ldots, y_{N-1}$. State a similar factorization of the DFT matrix as in Theorem 4.4, but this time where the matrix has 3×3 blocks.

c. Assume that $N = 3^n$, and that you implement the FFT using the formulas you have deduced in a. and b.. How many multiplications does this algorithm require?

d. Sketch a general procedure for speeding up the computation of the DFT, which uses the factorization of N into a product of prime numbers.

4.2 Efficient implementations of the DCT

In the preceding section we defined the DCT by expressing it in terms of the DFT. In particular, we can apply efficient implementations of the DFT, which we will shortly look at. However, the way we have defined the DCT, there is a penalty in that we need to compute a DFT of twice the length. We are also forced to use complex arithmetic (note that any complex multiplication corresponds to 4 real multiplications, and that any complex addition corresponds to 2 real additions). Is there a way to get around these penalties, so that we can get an implementation of the DCT which is more efficient, and uses less additions and multiplications than the one you made in Exercise 1? The following theorem states an expression of the DCT which achieves this. This expression is, together with a similar result for the DFT in the next section, much used in practical implementations:

Theorem 4.8 (DCT algorithm)**.** Let $\boldsymbol{y} = D_N \boldsymbol{x}$ be the N-point DCT of the vector $\boldsymbol{x}$. Then we have that

$$y_n = c_{n,N} \left(\cos\left(\pi \frac{n}{2N}\right) \Re((F_N \boldsymbol{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((F_N \boldsymbol{x}^{(1)})_n) \right), \quad (4.7)$$

where $c_{0,N} = 1$ and $c_{n,N} = \sqrt{2}$ for $n \geq 1$, and where $\boldsymbol{x}^{(1)} \in \mathbb{R}^N$ is defined by

$$(\boldsymbol{x}^{(1)})_k = x_{2k} \text{ for } 0 \leq k \leq N/2 - 1$$
$$(\boldsymbol{x}^{(1)})_{N-k-1} = x_{2k+1} \text{ for } 0 \leq k \leq N/2 - 1,$$

Proof. The N-point DCT of $\boldsymbol{x}$ is

$$y_n = d_{n,N} \sum_{k=0}^{N-1} x_k \cos\left(2\pi \frac{n}{2N}\left(k + \frac{1}{2}\right)\right).$$

Splitting this sum into two sums, where the indices are even and odd, we get

$$\begin{aligned} y_n = d_{n,N} &\sum_{k=0}^{N/2-1} x_{2k} \cos\left(2\pi \frac{n}{2N}\left(2k + \frac{1}{2}\right)\right) \\ &+ d_{n,N} \sum_{k=0}^{N/2-1} x_{2k+1} \cos\left(2\pi \frac{n}{2N}\left(2k + 1 + \frac{1}{2}\right)\right). \end{aligned}$$

If we reverse the indices in the second sum, this sum becomes

$$d_{n,N} \sum_{k=0}^{N/2-1} x_{N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(N - 2k - 1 + \frac{1}{2}\right)\right).$$

If we then also shift the indices with $N/2$ in this sum, we get

$$\begin{aligned} &d_{n,N} \sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(2N - 2k - 1 + \frac{1}{2}\right)\right) \\ &= d_{n,N} \sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi \frac{n}{2N}\left(2k + \frac{1}{2}\right)\right), \end{aligned}$$

where we used that cos is symmetric and periodic with period 2π. We see that we now have the same cos-terms in the two sums. If we thus define the vector

$\boldsymbol{x}^{(1)}$ as in the text of the theorem, we see that we can write

$$\begin{aligned}
y_n &= d_{n,N} \sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k \cos\left(2\pi \frac{n}{2N}\left(2k+\frac{1}{2}\right)\right) \\
&= d_{n,N} \Re\left(\sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k e^{-2\pi i n\left(2k+\frac{1}{2}\right)/(2N)}\right) \\
&= \sqrt{N} d_{n,N} \Re\left(e^{-\pi i n/(2N)} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} (\boldsymbol{x}^{(1)})_k e^{-2\pi i n k/N}\right) \\
&= c_{n,N} \Re\left(e^{-\pi i n/(2N)} (F_N \boldsymbol{x}^{(1)})_n\right) \\
&= c_{n,N} \left(\cos\left(\pi \frac{n}{2N}\right) \Re((F_N \boldsymbol{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((F_N \boldsymbol{x}^{(1)})_n)\right),
\end{aligned}$$

where we have recognized the N-point DFT, and where $c_{n,N} = \sqrt{N} d_{n,N}$. Inserting the values for $d_{n,N}$, we see that $c_{0,N} = 1$ and $c_{n,N} = \sqrt{2}$ for $n \geq 1$, which agrees with the definition of $c_{n,N}$ in the theorem. This completes the proof. □

With the result above we have avoided computing a DFT of double size. If we in the proof above define the $N \times N$-diagonal matrix Q_N by $Q_{n,n} = c_{n,N} e^{-\pi i n/(2N)}$, the result can also be written on the more compact form

$$\boldsymbol{y} = D_N \boldsymbol{x} = \Re\left(Q_N F_N \boldsymbol{x}^{(1)}\right).$$

We will, however, not use this form, since there is complex arithmetic involved, contrary to (4.7). Let us see how we can use (4.7) to implement the DCT, once we already have implemented the DFT in terms of the function `FFTImpl` as in Section 4.1:

```
function y = DCTImpl(x)
N = length(x);
if N == 1
    y = x;
else
    x1 = [x(1:2:(N-1)); x(N:(-2):2)];
    y = FFTImpl(x1);
    rp = real(y);
    ip = imag(y);
    y = cos(pi*((0:(N-1))')/(2*N)).*rp + sin(pi*((0:(N-1))')/(2*N)).*ip;
    y(2:N) = sqrt(2)*y(2:N);
end
```

In the code, the vector $\boldsymbol{x}^{(1)}$ is created first by rearranging the components, and it is sent as input to `FFTImpl`. After this we take real parts and imaginary parts, and multiply with the cos- and sin-terms in (4.7).

4.2.1 Efficient implementations of the IDCT

As with the FFT, it is straightforward to modify the DCT implementation so that it returns the IDCT. To see how we can do this, write from Theorem 4.8, for $n \geq 1$

$$\begin{aligned} y_n &= c_{n,N}\left(\cos\left(\pi\frac{n}{2N}\right)\Re((F_N\boldsymbol{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right)\Im((F_N\boldsymbol{x}^{(1)})_n)\right) \\ y_{N-n} &= c_{N-n,N}\left(\cos\left(\pi\frac{N-n}{2N}\right)\Re((F_N\boldsymbol{x}^{(1)})_{N-n}) + \sin\left(\pi\frac{N-n}{2N}\right)\Im((F_N\boldsymbol{x}^{(1)})_{N-n})\right) \\ &= c_{n,N}\left(\sin\left(\pi\frac{n}{2N}\right)\Re((F_N\boldsymbol{x}^{(1)})_n) - \cos\left(\pi\frac{n}{2N}\right)\Im((F_N\boldsymbol{x}^{(1)})_n)\right), \end{aligned}$$

where we have used the symmetry of F_N for real signals. These two equations enable us to determine $\Re((F_N\boldsymbol{x}^{(1)})_n)$ and $\Im((F_N\boldsymbol{x}^{(1)})_n)$ from y_n and y_{N-n}. We get

$$\begin{aligned} \cos\left(\pi\frac{n}{2N}\right)y_n + \sin\left(\pi\frac{n}{2N}\right)y_{N-n} &= c_{n,N}\Re((F_N\boldsymbol{x}^{(1)})_n) \\ \sin\left(\pi\frac{n}{2N}\right)y_n - \cos\left(\pi\frac{n}{2N}\right)y_{N-n} &= c_{n,N}\Im((F_N\boldsymbol{x}^{(1)})_n). \end{aligned}$$

Adding we get

$$\begin{aligned} c_{n,N}(F_N\boldsymbol{x}^{(1)})_n =& \cos\left(\pi\frac{n}{2N}\right)y_n + \sin\left(\pi\frac{n}{2N}\right)y_{N-n} + i(\sin\left(\pi\frac{n}{2N}\right)y_n - \cos\left(\pi\frac{n}{2N}\right)y_{N-n}) \\ =& (\cos\left(\pi\frac{n}{2N}\right) + i\sin\left(\pi\frac{n}{2N}\right))(y_n - iy_{N-n}) = e^{\pi in/(2N)}(y_n - iy_{N-n}). \end{aligned}$$

This means that $(F_N\boldsymbol{x}^{(1)})_n = \frac{1}{c_{n,N}}e^{\pi in/(2N)}(y_n - iy_{N-n})$ for $n \geq 1$. For $n = 0$, since $\Im((F_N\boldsymbol{x}^{(1)})_n) = 0$ we have that $(F_N\boldsymbol{x}^{(1)})_0 = \frac{1}{c_{0,N}}y_0$. This means that $\boldsymbol{x}^{(1)}$ can be recovered by taking the IDFT of the vector with component 0 being $\frac{1}{c_{0,N}}y_0 = y_0$, and the remaining components being $\frac{1}{c_{n,N}}e^{\pi in/(2N)}(y_n - iy_{N-n})$:

Theorem 4.9 (IDCT algorithm)**.** Let $\boldsymbol{x} = (D_N)^T\boldsymbol{y}$ be the IDCT of $\boldsymbol{y}$. and let $\boldsymbol{z}$ be the vector with component 0 being $\frac{1}{c_{0,N}}y_0$, and the remaining components being $\frac{1}{c_{n,N}}e^{\pi in/(2N)}(y_n - iy_{N-n})$. Then we have that

$$\boldsymbol{x}^{(1)} = (F_N)^H\boldsymbol{z},$$

where $\boldsymbol{x}^{(1)}$ is defined as in Theorem 4.8.

The implementation of IDCT can thus go as follows:

```
function x = IDCTImpl(y)
N = length(y);
if N == 1
    x = y(1);
```

```
else
  Q=exp(pi*1i*((0:(N-1))')/(2*N));
  Q(2:N)=Q(2:N)/sqrt(2);
  yrev=y(N:(-1):2);
  toapply=[ y(1); Q(2:N).*(y(2:N)-1i*yrev) ];
  x1=IFFTImpl(toapply);
  x=zeros(N,1);
  x(1:2:(N-1))=x1(1:(N/2));
  x(2:2:N)=x1(N:(-1):(N/2+1));
end
```

MATLAB also has a function for computing the DCT and IDCT, called `dct`, and `idct`. These functions are defined in MATLAB exactly as they are here, contrary to the case for the FFT.

4.2.2 Reduction in the number of multiplications with the DCT

Let us also state a result which confirms that the DCT and IDCT implementations we have described give the same type of reductions in the number multiplications as the FFT and IFFT:

Theorem 4.10 (Number of multiplications required by the DCT and IDCT algorithms)**.** Both the N-point DCT and IDCT factorizations given by Theorem 4.8 and Theorem 4.9 require $O(2(N+1)\log_2 N)$ real multiplications. In comparison, the number of real multiplications required by a direct implementation of the N-point DCT and IDCT is N^2.

Proof. By Theorem 4.7, the number of multiplications required by the FFT is $O(2N\log_2 N)$. By Theorem 4.8, two additional multiplications are needed for each index, giving additionally $2N$ multiplications in total, so that we end up with $O(2(N+1)\log_2 N)$ real multiplications. For the IDCT, note first that the vector $\boldsymbol{z} = \frac{1}{c_{n,N}} e^{\pi i n/(2N)}(y_n - i y_{N-n})$ seen in Theorem 4.9 should require $4N$ real multiplications to compute. But since the IDFT of $\boldsymbol{z}$ is real, $\boldsymbol{z}$ must have conjugate symmetry between the first half and the second half of the coefficients, so that we only need to perform $2N$ multiplications. Since the IFFT takes an additional $O(2N\log_2 N)$ real multiplications, we end up with a total of $O(2N + 2N\log_2 N) = O(2(N+1)\log_2 N)$ real multiplications also here. It is clear that the direct implementation of the DCT and IDCT needs N^2 multiplications, since only real arithmetic is involved. □

Since the DCT and IDCT can be implemented using the FFT and IFFT, it has the same advantages as the FFT when it comes to parallel computing. Much literature is devoted to reducing the number of multiplications in the DFT and the DCT even further than what we have done. In the next section

we will show an example on how this can be achieved, with the help of extra work and some tedious math. Some more notes on computational complexity are in order. For instance, we have not counted the operations sin and cos in the DCT. The reason is that these values can be precomputed, since we take the sine and cosine of a specific set of values for each DCT or DFT of a given size. This is contrary to to multiplication and addition, since these include the input values, which are only known at runtime. We have, however, not written down that we use precomputed arrays for sine and cosine in our algorithms: This is an issue to include in more optimized algorithms. Another point has to do with multiplication of $\frac{1}{\sqrt{N}}$. As long as $N = 2^{2r}$, multiplication with N need not be considered as a multiplication, since it can be implemented using a bitshift.

4.2.3 *An efficient joint implementation of the DCT and the FFT

We will now present a more advanced FFT algorithm, which will turn out to decrease the number of multiplications and additions even further. It also has the advantage that it avoids complex number arithmetic altogether (contrary to Theorem 4.1), and that it factors the computation into smaller FFTs and DCTs so that we can also use our previous DCT implementation. This implementation of the DCT and the DFT is what is mostly used in practice. For simplicity we will drop this presentation for the inverse transforms, and concentrate only on the DFT and the DCT.

Theorem 4.11 (Revised FFT algorithm)**.** Let $\boldsymbol{y} = F_N \boldsymbol{x}$ be the N-point DFT of the real vector $\boldsymbol{x}$. Then we have that

$$\Re(y_n) = \begin{cases} \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{z})_n & 0 \leq n \leq N/4-1 \\ \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) & n = N/4 \\ \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n) - (ED_{N/4}\boldsymbol{z})_{N/2-n} & N/4+1 \leq n \leq N/2-1 \end{cases} \tag{4.8}$$

$$\Im(y_n) = \begin{cases} \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) & q = 0 \\ \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{w})_{N/4-n} & 1 \leq n \leq N/4-1 \\ \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n) + (ED_{N/4}\boldsymbol{w})_{n-N/4} & N/4 \leq n \leq N/2-1 \end{cases} \tag{4.9}$$

where $\boldsymbol{x}^{(e)}$ is as defined in Theorem 4.1, where $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^{N/4}$ defined by

$$\begin{aligned} z_k &= x_{2k+1} + x_{N-2k-1} && 0 \leq k \leq N/4-1, \\ w_k &= (-1)^k(x_{N-2k-1} - x_{2k+1}) && 0 \leq k \leq N/4-1, \end{aligned}$$

and where E is a diagonal matrix with diagonal entries $E_{0,0} = \frac{1}{2}$ and $E_{n,n} = \frac{1}{2\sqrt{2}}$ for $n \geq 1$.

Proof. Taking real and imaginary parts in (4.1) we obtain

$$\Re(y_n) = \frac{1}{\sqrt{2}}\Re((F_{N/2}\boldsymbol{x}^{(e)})_n + \frac{1}{\sqrt{2}}\Re((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n)$$
$$\Im(y_n) = \frac{1}{\sqrt{2}}\Im((F_{N/2}\boldsymbol{x}^{(e)})_n + \frac{1}{\sqrt{2}}\Im((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n).$$

These equations explain the first parts on the right hand side in (4.8) and (4.9). Furthermore, for $0 \leq n \leq N/4-1$ we can write

$$\begin{aligned}
&\Re((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n) \\
&= \frac{1}{\sqrt{N/2}}\Re(e^{-2\pi in/N}\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi ink/(N/2)}) \\
&= \frac{1}{\sqrt{N/2}}\Re(\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi in(k+\frac{1}{2})/(N/2)}) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&\quad + \frac{1}{\sqrt{N/2}}\sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_{N/2-1-k} \cos\left(2\pi\frac{n(N/2-1-k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{N/4}}\sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_k + (\boldsymbol{x}^{(o)})_{N/2-1-k}) \cos\left(2\pi\frac{n\left(k+\frac{1}{2}\right)}{N/2}\right) \\
&= (E_0 D_{N/4}\boldsymbol{z})_n,
\end{aligned}$$

where we have used that cos is periodic with period 2π and symmetric, where z is the vector defined in the text of the theorem, where we have recognized the DCT matrix, and where E_0 is a diagonal matrix with diagonal entries $(E_0)_{0,0} = \frac{1}{\sqrt{2}}$ and $(E_0)_{n,n} = \frac{1}{2}$ for $n \geq 1$ (E_0 absorbs the factor $\frac{1}{\sqrt{N/2}}$, and the factor $d_{n,N}$ from the DCT). By absorbing the additional factor $\frac{1}{\sqrt{2}}$, we get a matrix E as stated in the theorem. For $N/4+1 \leq n \leq N/2-1$, everything above but the last statement is valid. We can now use that

$$\cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) = -\cos\left(2\pi\frac{\left(\frac{N}{2}-n\right)\left(k+\frac{1}{2}\right)}{N/2}\right)$$

to arrive at $-(E_0 D_{N/4}\boldsymbol{z})_{N/2-n}$ instead. For the case $n = \frac{N}{4}$ all the cosine entries are zero, and this completes (4.8). For the imaginary part, we obtain as

above

$$\begin{aligned}
&\Im((D_{N/2}F_{N/2}\boldsymbol{x}^{(o)})_n) \\
&= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} ((\boldsymbol{x}^{(o)})_{N/2-1-k} - (\boldsymbol{x}^{(o)})_k) \sin\left(2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/4-1} ((\boldsymbol{x}^{(o)})_{N/2-1-k} - (\boldsymbol{x}^{(o)})_k)(-1)^k \cos\left(2\pi \frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right).
\end{aligned}$$

where we have used that sin is periodic with period 2π and anti-symmetric, that

$$\begin{aligned}
\sin\left(2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) &= \cos\left(\frac{\pi}{2} - 2\pi \frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \cos\left(2\pi \frac{(N/4-n)(k+\frac{1}{2})}{N/2} - k\pi\right) \\
&= (-1)^k \cos\left(2\pi \frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right),
\end{aligned}$$

When $n = 0$ this is 0 since all the cosines entries are zero. When $1 \leq n \leq N/4$ this is $(E_0 D_{N/4}\boldsymbol{w})_{N/4-n}$, where $\boldsymbol{w}$ is the vector defined as in the text of the theorem. For $N/4 \leq n \leq N/2 - 1$ we arrive instead at $(E_0 D_{N/4}\boldsymbol{z})_{n-N/4}$, similarly to as above. This also proves (4.9), and the proof is done. □

As for Theorem 4.1, this theorem says nothing about the coefficients y_n for $n > \frac{N}{2}$. These are obtained in the same way as before through symmetry. The theorem also says nothing about $y_{N/2}$. This can be obtained with the same formula as in Theorem 4.1.

It is more difficult to obtain a matrix interpretation for Theorem 4.11, so we will only sketch an algorithm which implements it. The following code implements the recursive formulas for $\Re F_N$ and $\Im F_N$ in the theorem:

```
function y = FFTImpl2(x)
N = length(x);
if N == 1
    y = x;
elseif N==2
    y = 1/sqrt(2)*[x(1) + x(2); x(1) - x(2)];
else
    xe = x(1:2:(N-1));
    xo = x(2:2:N);
    yx = FFTImpl2(xe);

    z = x(N:(-2):(N/2+2))+x(2:2:(N/2));
    dctz = DCTImpl(z);
    dctz(1)=dctz(1)/2;
```

```
    dctz(2:length(dctz)) = dctz(2:length(dctz))/(2*sqrt(2));

    w = (-1).^((0:(N/4-1))').*(x(N:-2:(N/2+2))-x(2:2:(N/2)));
    dctw = DCTImpl(w);
    dctw(1)=dctw(1)/2;
    dctw(2:length(dctw)) = dctw(2:length(dctw))/(2*sqrt(2));

    y = yx/sqrt(2);
    y(1:(N/4))=y(1:(N/4))+dctz;
    if (N>4)
      y((N/4+2):(N/2))=y((N/4+2):(N/2))-dctz((N/4):(-1):2);
      y(2:(N/4))=y(2:(N/4))+1j*dctw((N/4):(-1):2);
    end
    y((N/4+1):(N/2))=y((N/4+1):(N/2))+1j*dctw;
    y = [y; ...
         sum(xe-xo)/sqrt(N); ...
         conj(y((N/2):(-1):2))];
end
```

In addition, we need to change the code for `DCTImpl` so that it calls `FFTImpl2` instead of `FFTImpl`. The following can now be shown:

Theorem 4.12 (Number of multiplications required by the revised FFT algorithm)**.** Let M_N be the number of real multiplications required by the revised algorithm of Theorem 4.11. Then we have that $M_N = O(\frac{2}{3}N \log_2 N)$.

This is a big reduction from the $O(2N \log_2 N)$ required by the FFT algorithm from Theorem 4.1. We will not prove Theorem 4.12. Instead we will go through the steps in a proof in Exercise 3. The revised FFT has yet a bigger advantage that the FFT when it comes to parallel computing: It splits the computation into, not two FFT computations, but three computations (one of which is an FFT, the other two DCT's). This makes it even easier to make use of many cores on computers which have support for this.

Exercises for Section 4.2

Ex. 1 — Write a function

```
function samples=DCTImpl(x)
```

which returns the DCT of the column vector $\boldsymbol{x} \in \mathbb{R}^{2N}$ as a column vector. The function should use the FFT-implementation from the previous section, and the factorization $C = E^{-1}AFB$ from above. The function should not construct the matrices A, B, E explicitly.

Ex. 2 — Explain why, if `FFTImpl` needs M_N multiplications A_N additions, then the number of multiplications and additions required by `DCTImpl` are $M_N + 2N$ and $A_N + N$, respectively.

Ex. 3 — In this exercise we will compute the number of real multiplications needed by the revised N-point FFT algorithm of Theorem 4.11, denoted M_N.

a. Explain from the algorithm of Theorem 4.11 that

$$M_N = 2(M_{N/4} + N/2) + M_{N/2} = N + M_{N/2} + 2M_{N/4}. \tag{4.10}$$

b. Explain why $x_r = M_{2^r}$ is the solution to the difference equation

$$x_{r+2} - x_{r+1} - 2x_r = 4 \times 2^r.$$

c. Show that the general solution to the difference equation is

$$x_r = \frac{2}{3} r 2^r + C 2^r + D(-1)^r.$$

d. Explain why $M_N = O(\frac{2}{3} N \log_2 N)$ (you do not need to write down the initial conditions for the difference equation in order to find the particular solution to it).

4.3 Summary

We obtained an implementation of the DFT which is more efficient in terms of the number of arithmetic operations than a direct implementation of the DFT. We also showed that this could be used for obtaining an efficient implementation of the DCT.