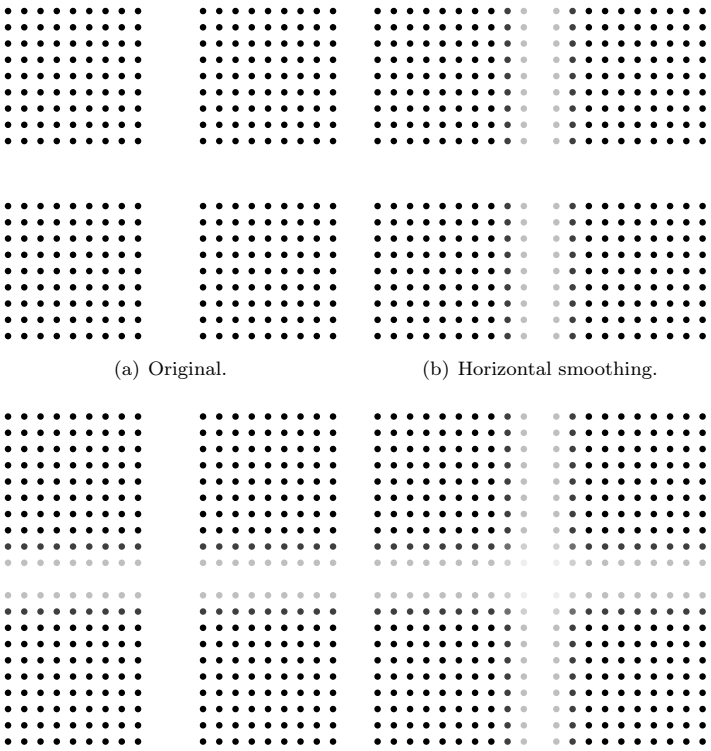
Chapter 7

Definition and properties of tensor products

The DFT, the DCT, and the wavelet transform were all defined as changes of basis for vectors or functions of one variable and therefore cannot be directly applied to higher dimensional data like images. In this chapter we will introduce a simple recipe for extending such one-dimensional schemes to two (and higher) dimensions. The basic ingredient is the *tensor product* construction. This is a general tool for constructing two-dimensional functions and filters from one-dimensional counterparts. This will allow us to generalise the filtering and compression techniques for audio to images, and we will also recognise some of the basic operations for images introduced in Chapter 6 as tensor product constructions.

A two-dimensional discrete function on a rectangular domain, like for example an image, is conveniently represented in terms of a matrix X with elements $X_{i,j}$, and with indices in the ranges $0 \leq i \leq M-1$ and $0 \leq j \leq N-1$. One way to apply filters to X would be to rearrange the matrix into a long vector, column by column. We could then apply a one-dimensional filter to this vector and then split the resulting vector into columns that can be reassembled back into a matrix again. This approach may have some undesirable effects near the boundaries between columns. In addition, the resulting computations may be rather ineffective. Consider for example the case where X is an $N \times N$ matrix so that the long vector has length N^2. Then a linear transformation applied to X involves multiplication with an $N^2 \times N^2$-matrix. Each such matrix multiplication may require as many as N^4 multiplications which is substantial when N is large.

The concept of tensor products can be used to address these problems. Using tensor products, one can construct operations on two-dimensional functions which inherit properties of one-dimensional operations. Tensor products also turn out to be computationally efficient.



(a) Original. (b) Horizontal smoothing.

(c) Vertical smoothing. (d) Horizontal and vertical smoothing.

Figure 7.1: The results of smoothing an image with the filter $\{1/4, \underline{1/2}, 1/4\}$ horizontally, vertically, and both. The pixels are shown as disks with intensity corresponding to the pixel values.

7.1 The tensor product of vectors

In Chapter 6 we saw examples of several filters applied to images. The filters of special interest to us now are those that determined a new image by combining neighbouring pixels, like the smoothing filter in Example 6.16 and the edge detection filter in Example 6.18. Our aim now is to try and apply filters like these as a repeated application of one-dimensional filters rather than using a computational molecule like in Chapter 6. It will be instructive to do this with an example before we describe the general construction, so let us revisit Example 6.16.

Figure 7.1 (a) shows an example of a simple image. We want to smooth this image X with the one-dimensional filter T given by $y_n = (T(\boldsymbol{x}))_n = (x_{n-1} + 2x_n + x_{n+1})/4$, or equivalently $T = \{1/4, \underline{1/2}, 1/4\}$. There are two obvious

one-dimensional operations we can do:

1. Apply the filter to each row in the image.
2. Apply the filter to each column in the image.

The problem is of course that these two operations will only smooth the image either horizontally or vertically as can be seen in the image in (b) and (c) of Figure 7.1.

So what else can we do? We can of course first smooth all the rows of the image and then smooth the columns of the resulting image. The result of this is shown in Figure 7.1 (d). Note that we could have performed the operations in the opposite order: first vertical smoothing and then horizontal smoothing, and currently we do not know if this is the same. We will show that these things actually are the same, and that computational molecules, as we saw in Chapter 6, naturally describe operations which are performed both vertically and horizontally. The main ingredient in this will be the tensor product construction. We start by defining the tensor product of two vectors.

Definition 7.1 (Tensor product of vectors)**.** If $\boldsymbol{x}, \boldsymbol{y}$ are vectors of length M and N, respectively, their tensor product $\boldsymbol{x} \otimes \boldsymbol{y}$ is defined as the $M \times N$-matrix defined by $(\boldsymbol{x} \otimes \boldsymbol{y})_{ij} = x_i y_j$. In other words, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.

In particular $\boldsymbol{x} \otimes \boldsymbol{y}$ is a matrix of rank 1, which means that most matrices cannot be written as tensor products. The special case $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$ is the matrix which is 1 at (i, j) and 0 elsewhere, and the set of all such matrices forms a basis for the set of $M \times N$-matrices.

Observation 7.2 (Interpretation of tensor products for vectors)**.** Let

$$\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1} \quad \text{and} \quad \mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$$

be the standard bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. Then

$$\mathcal{E}_{M,N} = \{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

is a basis for $L_{M,N}(\mathbb{R})$, the set of $M \times N$-matrices. This basis is often referred to as the standard basis for $L_{M,N}(\mathbb{R})$.

An image can simply be thought of as a matrix in $L_{M,N}(\mathbb{R})$. With this definition of tensor products, we can define operations on images by extending the one-dimensional filtering operations defined earlier.

Definition 7.3 (Tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, we define the linear mapping $S \otimes T : L_{M,N}(\mathbb{R}) \to L_{M,N}(\mathbb{R})$ by linear extension of $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. The linear mapping $S \otimes T$ is called the tensor product of the matrices S and T.

A couple of remarks are in order. First, from linear algebra we know that, when T is linear mapping from V and $T(\boldsymbol{v}_i)$ is known for a basis $\{\boldsymbol{v}_i\}_i$ of V, T is uniquely determined. In particular, since the $\{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{i,j}$ form a basis, there exists a unique linear transformation $S \otimes T$ so that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. This unique linear transformation is what we call the linear extension from the values in the given basis.

Secondly $S \otimes T$ also satisfies $(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) = (S\boldsymbol{x}) \otimes (T\boldsymbol{y})$. This follows from

$$\begin{aligned}(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) &= (S \otimes T)((\sum_i x_i \boldsymbol{e}_i) \otimes (\sum_j y_j \boldsymbol{e}_j)) = (S \otimes T)(\sum_{i,j} x_i y_j (\boldsymbol{e}_i \otimes \boldsymbol{e}_j)) \\ &= \sum_{i,j} x_i y_j (S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = \sum_{i,j} x_i y_j (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j) \\ &= \sum_{i,j} x_i y_j S\boldsymbol{e}_i ((T\boldsymbol{e}_j))^T = S(\sum_i x_i \boldsymbol{e}_i)(T(\sum_j y_j \boldsymbol{e}_j))^T \\ &= S\boldsymbol{x}(T\boldsymbol{y})^T = (S\boldsymbol{x}) \otimes (T\boldsymbol{y}).\end{aligned}$$

Here we used the result from Exercise 5. Linear extension is necessary anyway, since only rank 1 matrices have the form $\boldsymbol{x} \otimes \boldsymbol{y}$.

Example 7.4 (Smoothing an image)**.** Assume that S and T are both filters, and that $S = T = \{1/4, 1/2, 1/4\}$. Let us set $M = 5$ and $N = 7$, and let us compute $(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3)$. We have that

$$(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3) = (S\boldsymbol{e}_2)(T\boldsymbol{e}_3)^T = (\mathrm{col}_2 S)(\mathrm{col}_3 T)^T.$$

Since

$$S = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 1 & 2 \end{pmatrix} \qquad T = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix},$$

we find that

$$\frac{1}{4}\begin{pmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \end{pmatrix} = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 4 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We recognize here the computational molecule from Example 6.16 for smoothing an image. More generally it is not hard to see that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)$ is the matrix where the same computational molecule is placed with its centre at (i, j).

Clearly then, the linear extension $S \otimes T$ is obtained by placing the computational molecule over all indices, multiplying by the value at that index, and summing everything together. This is equivalent to the procedure for smoothing we learnt in Example 6.16. One can also write down component formulas for this as well. To achieve this, one starts with writing out the operation for tensor products of vectors:

$$\begin{aligned}
&((T \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}))_{i,j} \\
&= ((T\boldsymbol{x}) \otimes (T\boldsymbol{y}))_{i,j} = (T\boldsymbol{x})(T\boldsymbol{y})^T)_{i,j} = (T\boldsymbol{x})_i (T\boldsymbol{y})_j \\
&= \frac{1}{4}(x_{i-1} + 2x_i + x_{i+1})\frac{1}{4}(y_{j-1} + 2y_j + y_{j+1}) \\
&= \frac{1}{16}(x_{i-1}y_{j-1} + 2x_{i-1}y_j + x_{i-1}y_{j+1} \\
&\quad + 2x_i y_{j-1} + 4x_i y_j + 2x_i y_{j+1} + x_{i+1}y_{j-1} + 2x_{i+1}y_j + x_{i+1}y_{j+1}) \\
&= \frac{1}{16}((\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j-1} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j} + (\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j+1} \\
&\quad 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j-1} + 4(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j+1} \\
&\quad (\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j-1} + 2(\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j} + (\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j+1}).
\end{aligned}$$

Since this formula is valid when applied to any tensor product of two vectors, it is also valid when applied to any matrix:

$$\begin{aligned}
&((T \otimes T)X)_{i,j} \\
&= \frac{1}{16}(X_{i-1,j-1} + 2X_{i,j-1} + 2X_{i-1,j} + 4X_{i,j} + 2X_{i,j+1} \\
&\quad + 2X_{i+1,j} + X_{i+1,j+1})
\end{aligned}$$

This again confirms that the computational molecule given by Equation 6.3 in Example 6.16 is the tensor product of the filter $\{1/4, \underline{1/2}, 1/4\}$ with itself.

While we have seen that the computational molecules from Chapter 1 can be written as tensor products, not all computational molecules can be written as tensor products: we need of course that the molecule is a rank 1 matrix, since matrices which can be written as a tensor product always have rank 1.

The tensor product can be expressed explicitly in terms of matrix products.

Theorem 7.5. If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, the action of their tensor product on a matrix X is given by $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$.

Proof. We have that

$$\begin{aligned}
(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) &= (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j) \\
&= (\mathrm{col}_i(S)) \otimes (\mathrm{col}_j(T)) = \mathrm{col}_i(S)(\mathrm{col}_j(T))^T \\
&= \mathrm{col}_i(S)\mathrm{row}_j(T^T) = S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)T^T.
\end{aligned}$$

This means that $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$, since equality holds on the basis vectors $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$. □

This leads to the following implementation for the tensor product of matrices:

Theorem 7.6 (Implementation of a tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$, $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, and $X \in L_{M,N}(\mathbb{R})$, we have that $(S \otimes T)X$ can be computed as follows:

1. Apply S to every column of X.
2. Transpose the resulting matrix.
3. Apply T to every column in the resulting matrix.
4. Transpose the resulting matrix.

This recipe for computing $(S \otimes T)X$ is perhaps best seen if we write

$$(S \otimes T)X = SXT^T = (T(SX)^T)^T. \tag{7.1}$$

In the first step above we compute SX, in the second step $(SX)^T$, in the third step $T(SX)^T$, in the fourth step $(T(SX)^T)^T$. The reason for writing the tensor product this way, as an operation column by column, has to do with with that S and T are mostly filters for our purposes, and that we want to reuse efficient implementations instead of performing full matrix multiplications, just as we decided to express a wavelet transformation in terms of filters. The reason for using columns instead of rows has to do with that we have expressed filtering as a matrix by column multiplication. Note that this choice of using columns instead of rows should be influenced by how the computer actually stores values in a matrix. If these values are stored column by column, performing operations columnwise may be a good idea, since then the values from the matrix are read in the same order as they are stored. If matrix values are stored row by row, it may be a good idea to rewrite the procedure above so that operations are performed row by row also (see Exercise 7).

Theorem 7.6 leads to the following algorithm for computing the tensor product of matrices:

```
[M,N]=size(X);
for col=1:N
  X(:,col)=S*X(:,col);
end
X=X'
for col=1:M
  X(:,col)=T*X(:,col);
end
X=X';
```

This algorithm replaces the rows and columns in X at each step. In the following, $S = T$ in most cases. In this case we can replace with the following algorithm, which is even simpler:

```
for k=1:2
  for col=1:size(X,2)
    X(:,col)=S*X(:,col);
  end
  X=X';
end
```

In an efficient algorithm, we would of course replace the matrix multiplications with S and T with efficient implementations.

If we want to apply a sequence of tensor products of filters to a matrix, the order of the operations does not matter. This will follow from the next result:

Corollary 7.7. If $S_1 \otimes T_1$ and $S_2 \otimes T_2$ are two tensor products of one dimensional filters, then $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1 S_2) \otimes (T_1 T_2)$.

Proof. By Theorem 7.5 we have that

$$(S_1 \otimes T_1)(S_2 \otimes T_2)X = S_1(S_2 X T_2^T)T_1^T = (S_1 S_2)X(T_1 T_2)^T = ((S_1 S_2) \otimes (T_1 T_2))X.$$

for any $X \in L_{M,N}(\mathbb{R})$. This proves the result. □

Suppose that we want to apply the operation $S \otimes T$ to an image. We can write

$$S \otimes T = (S \otimes I)(I \otimes T) = (I \otimes T)(S \otimes I). \tag{7.2}$$

Moreover, from Theorem 7.5 it follows that

$$\begin{aligned}(S \otimes I)X &= SX \\ (I \otimes T)X &= XT^T = (TX^T)^T.\end{aligned}$$

This means that $S \otimes I$ corresponds to applying S to each column in X, and $I \otimes T$ corresponds to applying T to each row in X. When S and T are smoothing filters, this is what we refered to as vertical smoothing and horizontal smoothing, respectively. The relations in Equation (7.2) thus have the following interpretation (alternatively note that the order of left or right multiplication does not matter).

Observation 7.8. The order of vertical and horizontal smoothing does not matter, and any tensor product of filters $S \otimes T$ can be written as a horizontal filtering operation $I \otimes T$ followed by a vertical filtering operation $S \otimes I$.

In fact, the order of any vertical operation $S \otimes I$ and horizontal operation $I \otimes T$ does not matter: it is not required that the operations are filters. For filters we have a stronger result: If S_1, T_1, S_2, T_2 all are filters, we have from Corollary 7.7 that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_2 \otimes T_2)(S_1 \otimes T_1)$, since all filters commute. This does not hold in general since general matrices do not commute.

Example 7.9 (Detecting edges)**.** Consider the bass reducing filter $T = \{1/2, \underline{0}, -1/2\}$, i.e. $(T(\boldsymbol{x}))_n = \frac{1}{2}(x_{n+1} - x_{n-1})$. We compute the vertical filtering operation $T \otimes I$ as

$$\begin{aligned}&((T \otimes I)(\boldsymbol{x} \otimes \boldsymbol{y}))_{i,j}] = (T\boldsymbol{x})_i y_j \\ &= \frac{1}{2}(x_{i+1} - x_{i-1})y_j = \frac{1}{2}x_{i+1}y_j - \frac{1}{2}x_{i-1}y_j = \frac{1}{2}(\boldsymbol{x} \otimes \boldsymbol{y})_{i+1,j} - \frac{1}{2}(\boldsymbol{x} \otimes \boldsymbol{y})_{i-1,j}.\end{aligned}$$

This shows as above that $T \otimes I$ is the transformation where the computational molecule given by Equation 6.6 in Example 6.18 is placed over the image samples. This tensor product can thus be used for detecting vertical edges in images.

Exercises for Section 7.1

Ex. 1 — With $T = \{1/2, \underline{0}, -1/2\}$, show that $I \otimes T$ is the transformation where the computational molecule is given by Equation 6.7 in Example 6.18. This tensor product can thus be used for detecting horizontal edges in images.

Ex. 2 — With $T = \{1/2, \underline{0}, -1/2\}$, show that $T \otimes T$ corresponds to the computational molecule given by Equation 6.9 in Example 6.18.

Ex. 3 — Let T be the moving average filter of length $2L + 1$, i.e. $T = \frac{1}{L}\{\underbrace{1, \cdots, 1, \underline{1}, 1, \cdots, 1}_{2L+1 \text{ times}}\}$. As in Example 7.4, find the computational molecule of $T \otimes T$.

Ex. 4 — Verify that the computational molecule given by Equation 6.4 in Example 6.18 is the same as that of $T \otimes T$, where $T = \{\frac{1}{64}, \frac{6}{64}, \frac{15}{64}, \underline{\frac{20}{64}}, \frac{15}{64}, \frac{6}{64}, \frac{1}{64}\}$ (the coefficients come from row 6 of Pascals triangle).

Ex. 5 — Show that the mapping $F(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y}$ is bi-linear, i.e. that $F(\alpha\boldsymbol{x}_1 + \beta\boldsymbol{x}_2, \boldsymbol{y}) = \alpha F(\boldsymbol{x}_1, \boldsymbol{y}) + \beta F(\boldsymbol{x}_2, \boldsymbol{y})$, and $F(\boldsymbol{x}, \alpha\boldsymbol{y}_1 + \beta\boldsymbol{y}_2) = \alpha F(\boldsymbol{x}, \boldsymbol{y}_1) + \beta F(\boldsymbol{x}, \boldsymbol{y}_2)$.

Ex. 6 — Find matrices $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ so that the following mappings from $L_{M,N}(\mathbb{R})$ to $L_{M,N}(\mathbb{R})$ can be written on the form $X \to SXT^T = (S \otimes T)X$:

a. The mapping which reverses the order of the rows in a matrix.
b. The mapping which reverses the order of the columns in a matrix.
c. The mapping which transposes a matrix.

Ex. 7 — Find an alternative form for Equation (7.1) and an accompanying reimplementation of Theorem 7.6 which is adapted to the case when we want all operations to be performed row by row, instead of column by column.

7.2 Change of bases in tensor products

In this section we will prove a specialization of our previous result to the case where S and T are change of coordinate matrices. We start by proving the following:

Theorem 7.10. If $\mathcal{B}_1 = \{\boldsymbol{v}_i\}_{i=0}^{M-1}$ is a basis for $\mathbb{R}^M$, and $\mathcal{B}_2 = \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ is a basis for $\mathbb{R}^N$, then $\{\boldsymbol{v}_i \otimes \boldsymbol{w}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $L_{M,N}(\mathbb{R})$. We denote this basis by $\mathcal{B}_1 \otimes \mathcal{B}_2$.

Proof. Suppose that $\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = 0$. Setting $\boldsymbol{h}_i = \sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j$ we get

$$\sum_{j=0}^{N-1} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \boldsymbol{v}_i \otimes (\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j) = \boldsymbol{v}_i \otimes \boldsymbol{h}_i.$$

where we have used the bi-linearity of the tensor product mapping $(\boldsymbol{x}, \boldsymbol{y}) \to \boldsymbol{x} \otimes \boldsymbol{y}$ (Exercise 7.1.5). This means that

$$\boldsymbol{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \sum_{i=0}^{M-1} \boldsymbol{v}_i \otimes \boldsymbol{h}_i = \sum_{i=0}^{M-1} \boldsymbol{v}_i \boldsymbol{h}_i^T.$$

Column k in this matrix equation says $\boldsymbol{0} = \sum_{i=0}^{M-1} h_{i,k}\boldsymbol{v}_i$, where $h_{i,k}$ are the components in $\boldsymbol{h}_i$. By linear independence of the $\boldsymbol{v}_i$ we must have that $h_{0,k} = h_{1,k} = \cdots = h_{M-1,k} = 0$. Since this applies for all k, we must have that all $\boldsymbol{h}_i = 0$. This means that $\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j = \boldsymbol{0}$ for all i, from which it follows by linear independence of the $\boldsymbol{w}_j$ that $\alpha_{i,j} = 0$ for all j, and for all i. This means that $\mathcal{B}_1 \otimes \mathcal{B}_2$ is a basis. □

In particular, as we have already seen, the standard basis for $L_{M,N}(\mathbb{R})$ can be written $\mathcal{E}_{M,N} = \mathcal{E}_M \otimes \mathcal{E}_N$. This is the basis for a useful convention: For a tensor product the bases are most naturally indexed in two dimensions, rather than the usual sequential indexing. This difference translates also to the meaning of coordinate vectors, which now are more naturally thought of as coordinate matrices:

Definition 7.11 (Coordinate matrix)**.** Let $\{\boldsymbol{v}_i\}_{i=0}^{M-1}, \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ be bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. By the coordinate matrix of $\sum_{k,l} \alpha_{k,l}(\boldsymbol{v}_k \otimes \boldsymbol{w}_l)$ we will mean the $M \times N$-matrix X with entries $X_{kl} = \alpha_{k,l}$.

We will have use for the following theorem, which shows how change of coordinates in $\mathbb{R}^M$ and $\mathbb{R}^N$ translate to a change of coordinates in the tensor product:

Theorem 7.12 (Change of coordinates in tensor products)**.** Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for $\mathbb{R}^M$, and $\mathcal{B}_2, \mathcal{C}_2$ are bases for $\mathbb{R}^N$, and that S is the change of coordinates matrix from $\mathcal{C}_1$ to $\mathcal{B}_1$, and that T is the change of coordinates matrix from $\mathcal{C}_2$ to $\mathcal{B}_2$. Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $L_{M,N}(\mathbb{R})$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \tag{7.3}$$

Proof. Let $\boldsymbol{c}_{ki}$ be the i'th basis vector in $\mathcal{C}_k$, $\boldsymbol{b}_{ki}$ the i'th basis vector in $\mathcal{B}_k$, $k = 1, 2$. Since any change of coordinates is linear, it is enough to show that it coincides with $X \to SXT^T$ on the basis $\mathcal{C}_1 \otimes \mathcal{C}_2$. The basis vector $\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j}$ has coordinate vector $X = \boldsymbol{e}_i \otimes \boldsymbol{e}_j$ in $\mathcal{C}_1 \otimes \mathcal{C}_2$. With the mapping $X \to SXT^T$ this is sent to

$$SXT^T = S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)T^T = \mathrm{col}_i(S)\mathrm{row}_j(T^T).$$

On the other hand, since column i in S is the coordinates of $\boldsymbol{c}_{1i}$ in the basis $\mathcal{B}_1$, and column j in T is the coordinates of $\boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_2$, we can write

$$\begin{aligned}\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j} &= \left(\sum_k S_{k,i}\boldsymbol{b}_{1k}\right) \otimes \left(\sum_l T_{l,j}\boldsymbol{b}_{2l}\right) = \sum_{k,l} S_{k,i}T_{l,j}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) \\ &= \sum_{k,l} S_{k,i}(T^T)_{j,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) = \sum_{k,l} (\mathrm{col}_i(S)\mathrm{row}_j(T^T))_{k,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l})\end{aligned}$$

we see that the coordinate vector of $\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_1 \otimes \mathcal{B}_2$ is $\mathrm{col}_i(S)\mathrm{row}_j(T^T)$. In other words, change of coordinates coincides with $X \to SXT^T$, and the proof is done. □

In both cases of tensor products of matrices and change of coordinates in tensor products, we see that we need to compute the mapping $X \to SXT^T$. This means that we can restate Theorem 7.6 for change of coordinates as follows:

Theorem 7.13 (Implementation of change of coordinates in tensor products)**.** The change of coordinates from $\mathcal{C}_1 \otimes \mathcal{C}_2$ to $\mathcal{B}_1 \otimes \mathcal{B}_2$ can be implemented as follows:

1. For every column in the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, perform a change of coordinates from $\mathcal{C}_1$ to $\mathcal{B}_1$.
2. Transpose the resulting matrix.
3. For every column in the resulting matrix, perform a change of coordinates from $\mathcal{C}_2$ to $\mathcal{B}_2$.
4. Transpose the resulting matrix.

We can reuse the algorithm from the previous section to implement this. In the following operations on images, we will visualize the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

Example 7.14 (Change of coordinates with the DFT)**.** The DFT is one particular change of coordinates which we have considered. The DFT was the change of coordinates from the standard basis to the Fourier basis. The corresponding change of coordinates in a tensor product is obtained by substituting with the DFT as the function implementing change of coordinates in Theorem 7.13. The change of coordinates in the opposite direction is obtained by using the IDFT instead of the DFT.

Modern image standards do typically not apply a change of coordinates to the entire image. Rather one splits the image into smaller squares of appropriate size, called blocks, and perform change of coordinates independently for each block. With the JPEG standard, the blocks are always 8×8. It is of course not a coincidence that a power of 2 is chosen here, since the DFT takes a simplified form in case of powers of 2.

The DFT values express frequency components. The same applies for the two-dimensional DFT and thus for images, but frequencies are now represented in two different directions. The thing which actually provides compression in many image standards is that frequency components which are small are set to 0. This corresponds to neglecting frequencies in the image which have small contributions. This type of lossy compression has little effect on the human perception of the image, if we use a suitable neglection threshold.

In Figure 7.3 we have applied the two-dimensional DFT to our test image. We have then neglected DFT coefficients which are below certain thresholds, and transformed the samples back to reconstruct the image. When increasing the threshold, the image becomes more and more unclear, but the image is quite clear in the first case, where as much as more than 90% of the samples have been neglected. The blocking effect at the block boundaries is clearly visible.

Example 7.15 (Change of coordinates with the DCT)**.** Similarly to the DFT, the DCT was the change of coordinates from the standard basis to what we called the DCT basis. The DCT is used more than the DFT in image processing. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT in

(a) Threshold 30. 91.3% of the DFT-values were neglected

(b) Threshold 50. 94.9% of the DFT-values were neglected

(c) Threshold 100. 97.4% of the DFT-values were neglected

Figure 7.2: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold were neglected.

Theorem 7.13. The JPEG standard actually applies a two-dimensional DCT to the blocks of size 8×8, it does not apply the two-dimensional DFT.

If we follow the same strategy for the DCT as for the DFT, so that we neglect DCT-coefficients which are below a given threshold, we get the images shown in Figure 7.3. We see similar effects as with the DFT, but it seems that the latter images are a bit clearer, verifying that the DCT is a better choice than the DFT. It is also interesting to compare with what happens when we drop splitting the image into blocks. Of course, when we neglect many of the DCT-coefficients, we should see some artifacts, but there is no reason to believe that these should be at the old block boundaries. The new artifacts can be seen in Figure 7.4, where the same thresholds as before have been used. Clearly, the new artifacts take a completely different shape.

In the exercises you will be asked to implement functions which generate the images shown in these examples.

Exercises for Section 7.2

Ex. 1 — Implement functions

```
function newx=FFT2Impl(x)
function x=IFF2Impl(newx)
function newx=DCT2Impl(x)
function x=IDCT2Impl(newx)
```

which implement the two-dimensional DCT, FFT, and their inverses as described in this section. Base your code on the algorithm at the end of Section 7.1.

(a) Threshold 30. 93.9% of the DCT-values were neglected

(b) Threshold 50. 96.0% of the DCT-values were neglected

(c) Threshold 100. 97.6% of the DCT-values were neglected

Figure 7.3: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected.

(a) Threshold 30. 93.7% of the DCT-values were neglected

(b) Threshold 50. 96.7% of the DCT-values were neglected

(c) Threshold 100. 98.5% of the DCT-values were neglected

Figure 7.4: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected. The image has not been split into blocks here.

Ex. 2 — Implement functions

```
function samples=transform2jpeg(x)
function samples=transform2invjpeg(x)
```

which splits the image into blocks of size 8×8, and performs the DCT2/IDCT2 on each block. Finally run the code

```
function showDCThigher(threshold)
  img = double(imread('mm.gif','gif'));
  newimg=transform2jpeg(img);
  thresholdmatr=(abs(newimg)>=threshold);
  zeroedout=size(img,1)*size(img,2)-sum(sum(thresholdmatr));
  newimg=transform2invjpeg(newimg.*thresholdmatr);
  imageview(abs(newimg));
  fprintf('%i percent of samples zeroed out\n',...
      100*zeroedout/(size(img,1)*size(img,2)));
```

for different threshold parameters, and check that this reproduces the test images of this section, and prints the correct numbers of values which have been neglected (i.e. which are below the threshold) on screen.

7.3 Summary

We defined the tensor product, and saw how this could be used to define operations on images in a similar way to how we defined operations on sound. It turned out that the tensor product construction could be used to construct some of the operations on images we looked at in the previous chapter, which now could be factorized into first filtering the columns in the image, and then filtering the rows in the image. We went through an algorithm for computing the tensor product, and established how we could perform change of coordinates in tensor products. This enables us to define two-dimensional extensions of the DCT and the DFT and their inverses, and we used these extensions to experiment on images.