4 Neighborhood Operations

4.1 Basic Properties and Purpose

4.1.1 Object Recognition and Neighborhood Operations

An analysis of the spatial relations of the gray values in a small neighborhood provides the first clue for the recognition of objects in images. Let us take a scene containing objects with uniform radiance as a simple example. If the gray value does not change in a small neighborhood, the neighborhood lies within an object. If, however, the gray value changes significantly, an edge of an object crosses the neighborhood. In this way, we recognize areas of constant gray values and edges.

Just processing individual pixels in an image by *point operations* does not provide this type of information. In Chapter 10 we show in detail that such operations are only useful as an initial step of image processing to correct inhomogeneous and nonlinear responses of the imaging sensor, to interactively manipulate images for inspection, or to improve the visual appearance.

A new class of operations is necessary that combines the pixels of a small neighborhood in an appropriate manner and yields a result that forms a new image. Operations of this kind belong to the general class of *neighborhood operations*. These are the central tools for *low-level image processing*. This is why we discuss the possible classes of neighborhood operations and their properties in this chapter.

The result of any neighborhood operation is still an image. However, its content has been changed. A properly designed neighborhood operation to detect edges, for instance, should show bright values at pixels that belong to an edge of an object while all other pixels — independent of their gray value — should show low values. This example illustrates that by the application of a neighborhood operator, information is generally lost. We can no longer infer the original gray values. This is why neighborhood operations are also called *filters*. They extract a certain *feature* of interest from an image. The image resulting from a neighborhood operator is therefore also called a *feature image*.

It is obvious that operations combining neighboring pixels to form a new image can perform quite different image processing tasks:

- Detection of simple local structures such as edges, corners, lines, and areas of constant gray values (Chapters 12 and 13)

- Motion determination (Chapter 14)
- Texture analysis (Chapter 15)
- *Reconstruction* of images taken with indirect imaging techniques such as *tomography* (Chapter 17)
- *Restoration* of images degraded by defocusing, motion blur, or similar errors during image acquisition (Chapter 17)
- Correction of disturbances caused by errors in image acquisition or transmission. Such errors will result in incorrect gray values for a few individual pixels (Chapter 17)

4.1.2 General Definition

A neighborhood operator N takes the values of the neighborhood around a point, performs some operations with them, and writes the result back on the pixel. This operation is repeated for all points of the signal.

Definition 4.1 (Continuous neighborhood operator) *A continuous neighborhood operator maps a multidimensional continuous signal $g(\boldsymbol{x})$ onto itself by the following operation*

$$g'(\boldsymbol{x}) = N(\{g(\boldsymbol{x}')\}, \forall (\boldsymbol{x} - \boldsymbol{x}') \in \mathbb{M}) \qquad (4.1)$$

where $\mathbb{M}$ is a compact area.

The area $\mathbb{M}$ is called *mask*, *window*, *region of support*, or *structure element* of the neighborhood operation. For the computation of $g'(\boldsymbol{x})$, the size and shape of $\mathbb{M}$ determine the neighborhood operation by specifying the input values of g in the area $\mathbb{M}$ that is shifted with its origin to the point $\boldsymbol{x}$. The neighborhood operation N itself is not specified here. It can be of any type. For symmetry reasons the mask is often symmetric and has its origin in the symmetry center.

Definition 4.2 (Discrete neighborhood operator) *A discrete neighborhood operator maps an $M \times N$ matrix onto itself by the operation*

$$G'_{m,n} = N(G_{m'-m,n'-n}, \forall \left[m', n'\right]^T \in \mathbb{M}), \qquad (4.2)$$

where $\mathbb{M}$ is now a discrete set of points.

Expressions equivalent to Def. 4.2 can easily be written for dimensions other than two. Although Eqs. (4.1) and (4.2) do not specify in any way the type of neighborhood operation that is performed, they still reveal the common structure of all neighborhood operations.

4.1.3 Mask Size and Symmetry

The first characteristic of a neighborhood operation is the size of the neighborhood. The window may be rectangular or of any other form. We must also specify the position of the pixel relative to the window that will receive the result of the operation. With regard to symmetry, the most natural choice is to place the result of the operation at the pixel in the center of an odd-sized mask of the size $(2R+1) \times (2R+1)$.

Even-sized masks seem not to be suitable for neighborhood operations because there is no pixel that lies in the center of the mask. If the result of the neighborhood operation is simply written back to pixels that lie between the original pixels in the center of the mask, we can apply them nevertheless. Thus, the resulting image is shifted by half the pixel distance into every direction. Because of this shift, image features computed by even-sized masks should never be combined with original gray values because this would lead to considerable errors. If we apply several masks in parallel and combine the resulting feature images, all masks must be either even-sized or odd-sized into the same direction. Otherwise, the output lattices do not coincide.

4.1.4 Operator Notation

It is useful to introduce an *operator notation* for neighborhood operators. In this way, complex composite neighbor operations are easily comprehensible. All operators will be denoted by calligraphic letters, such as $\mathcal{B}, \mathcal{D}, \mathcal{H}, \mathcal{S}$. The operator $\mathcal{H}$ transforms the image $\boldsymbol{G}$ into the image $\boldsymbol{G}'$: $\boldsymbol{G}' = \mathcal{H}\boldsymbol{G}$. This notation can be used for continuous and discrete signals of any dimension and leads to a compact *representation-independent notation* of signal-processing operations.

Writing the operators one after the other denotes consecutive application. The rightmost operator is applied first. An exponent expresses consecutive application of the same operator

$$\underbrace{\mathcal{H}\mathcal{H}\ldots\mathcal{H}}_{\text{p times}} = \mathcal{H}^p. \tag{4.3}$$

If the operator acts on a single image, the operand, which is to the right in the equations, can be omitted. In this way, operator equations can be written without targets. Furthermore, we will use braces in the usual way to control the order of execution. We can write basic properties of operators in an easily comprehensible way, e. g.,

$$\begin{array}{ll} \text{commutativity} & \mathcal{H}_1\mathcal{H}_2 = \mathcal{H}_2\mathcal{H}_1 \\ \text{associativity} & \mathcal{H}_1(\mathcal{H}_2\mathcal{H}_3) = (\mathcal{H}_1\mathcal{H}_2)\mathcal{H}_3 \\ \text{distributivity over addition} & (\mathcal{H}_1+\mathcal{H}_2)\mathcal{H}_3 = \mathcal{H}_1\mathcal{H}_3 + \mathcal{H}_2\mathcal{H}_3 \end{array} \tag{4.4}$$

Other operations such as addition can also be used in this operator notation. Care must be taken, however, with any nonlinear operation. As soon as a nonlinear operator is involved, the order in which the operators are executed must strictly be given.

A simple example for a nonlinear operator is pointwise multiplication of images, a dyadic point operator. As this operator occurs frequently, it is denoted by a special symbol, a centered dot ($\cdot$). This symbol is required in order to distinguish it from successive application of operators. The operator expression $\mathcal{B}(\mathcal{D}_p \cdot \mathcal{D}_q)$, for instance, means: apply the operators $\mathcal{D}_p$ and $\mathcal{D}_q$ to the same image, multiply the result pointwise, and apply the operator $\mathcal{B}$ to the product image. Without parentheses the expression $\mathcal{B}\mathcal{D}_p \cdot \mathcal{D}_q$ would mean: apply the operator $\mathcal{D}_q$ to the image and apply the operator $\mathcal{D}_p$ and $\mathcal{B}$ to the same image and then multiply the results point by point. The used operator notation thus gives monadic operators precedence over dyadic operators. If required for clarity, a placeholder for an object onto which an operator is acting is used, denoted by the symbol ":". With a placeholder, the aforementioned operator combination is written as $\mathcal{B}(\mathcal{D}_p : \cdot \mathcal{D}_q :)$.

In the remainder of this chapter we will discuss the two most important classes of neighborhood operations, linear shift-invariant filters (Section 4.2) and rank value filters (Section 4.3). An extra section is devoted to a special subclass of linear-shift-invariant filters, known as recursive filters (Section 4.5).

4.2 Linear Shift-Invariant Filters

4.2.1 Discrete Convolution

First we focus on the question as to how we can combine the gray values of pixels in a small neighborhood.

The elementary combination of the pixels in the window is given by an operation which multiplies each pixel in the range of the filter mask with the corresponding weighting factor of the mask, adds up the products, and writes the sum to the position of the center pixel:

$$\boxed{\begin{aligned} g'_{mn} &= \sum_{m'=-r}^{r} \sum_{n'=-r}^{r} h_{m'n'} g_{m-m',n-n'} \\ &= \sum_{m''=-r}^{r} \sum_{n''=-r}^{r} h_{-m'',-n''} g_{m+m'',n+n''}. \end{aligned}} \tag{4.5}$$

In Section 2.3.4, the *discrete convolution* was defined in Eq. (2.55) as:

$$g'_{mn} = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} h_{m'n'} g_{m-m',n-n'} \tag{4.6}$$

Both definitions are equivalent if we consider the periodicity in the space domain given by Eq. (2.42). From Eq. (2.42) we infer that negative indices are equivalent to positive coefficients by the relations

$$g_{-n} = g_{N-n}, \quad g_{-n,-m} = g_{N-n,M-m}. \tag{4.7}$$

The restriction of the sum in Eq. (4.5) reflects the fact that the elements of the matrix $\boldsymbol{H}$ are zero outside the few points of the $(2R + 1) \times (2R + 1)$ filter mask. Thus the latter representation is much more practical and gives a better comprehension of the filter operation. For example, the following 3×3 filter mask and the $M \times N$ matrix $\boldsymbol{H}$ are equivalent

$$\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0_\bullet & -1 \\ 2 & 1 & 0 \end{bmatrix} \equiv \begin{bmatrix} 0_\bullet & -1 & 0 & \ldots & 0 & 1 \\ 1 & 0 & 0 & \ldots & 0 & 2 \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ -1 & -2 & 0 & \ldots & 0 & 0 \end{bmatrix}. \tag{4.8}$$

A W-dimensional filter operation can be written with a simplified vector indexing:

$$\boxed{g'_{\boldsymbol{n}} = \sum_{\boldsymbol{n}'=-\boldsymbol{R}}^{\boldsymbol{R}} h_{-\boldsymbol{n}'} g_{\boldsymbol{n}+\boldsymbol{n}'}} \tag{4.9}$$

with $\boldsymbol{n} = [n_1, n_2, \ldots, n_W]$, $\boldsymbol{R} = [R_1, R_2, \ldots, R_W]$, where $g_{\boldsymbol{n}}$ is an element of a W-dimensional signal $g_{n_1,n_2,\ldots,n_W}$. The notation for the sums in this equation is an abbreviation for

$$\sum_{\boldsymbol{n}'=-\boldsymbol{R}}^{\boldsymbol{R}} = \sum_{n'_1=-R_1}^{R_1} \sum_{n'_2=-R_2}^{R_2} \ldots \sum_{n'_W=-R_W}^{R_W}. \tag{4.10}$$

The vectorial indexing introduced here allows writing most of the relations for signals of arbitrary dimension in a simple way.

4.2.2 Symmetries

With regard to symmetry, we can distinguish two important classes of filters: even and odd filters with the condition in one or more directions that

$$h_{-m,n} = \pm h_{mn} \quad \text{or} \quad h_{m,-n} = \pm h_{mn}, \tag{4.11}$$

where the + and − signs stand for *even* and *odd* symmetry. From this definition we can immediately reduce Eq. (4.5) to make the computation of one-dimensional filters more efficient:

$$\begin{aligned}\text{even:}\quad g'_{mn} &= h_0 g_{m,n} + \sum_{n'=1}^{r} h_{n'}(g_{m,n-n'} + g_{m,n+n'})\\ \text{odd:}\quad g'_{mn} &= \sum_{n'=1}^{r} h_{n'}(g_{m,n-n'} - g_{m,n+n'}).\end{aligned} \tag{4.12}$$

The sums only run over half of the filter mask, excluding the center pixel, which must be treated separately because it has no symmetric counterpart. It can be omitted for the odd filter since the coefficient at the center pixel is zero according to Eq. (4.11).

In the 2-D case, the equations become more complex because it is now required to consider the symmetry in each direction separately. A 2-D filter with even symmetry in both directions reduces to

$$\begin{aligned} g'_{m,n} &= h_{00}g_{nm}\\ &+ \sum_{n'=1}^{r} h_{0n'}(g_{m,n-n'} + g_{m,n+n'})\\ &+ \sum_{m'=1}^{r} h_{m'0}(g_{m-m',n} + g_{m+m',n})\\ &+ \sum_{m'=1}^{r}\sum_{n'=1}^{r} h_{m'n'}(g_{m-m',n-n'} + g_{m-m',n+n'}\\ &\qquad + g_{m+m',n-n'} + g_{m+m',n+n'}).\end{aligned} \tag{4.13}$$

2-D filters can have different types of symmetries in different directions. For example, they can be odd in horizontal and even in vertical directions. Then

$$\begin{aligned} g'_{m,n} &= \sum_{n'=1}^{r} h_{0n'}(g_{m,n-n'} - g_{m,n+n'})\\ &+ \sum_{m'=1}^{r}\sum_{n'=1}^{r} h_{m'n'}(g_{m-m',n-n'} - g_{m-m',n+n'}\\ &\qquad + g_{m+m',n-n'} - g_{m+m',n+n'}).\end{aligned} \tag{4.14}$$

The equations for higher dimensions are even more complex [89].

4.2.3 Computation of Convolution

The discrete convolution operation is such an important operation that it is worth studying it in detail to see how it works. First, we might be confused by the negative signs of the indices m' and n' for either the mask or the image in Eq. (4.5). This just means that we reflect either the mask or the image at its symmetry center before we put the mask over

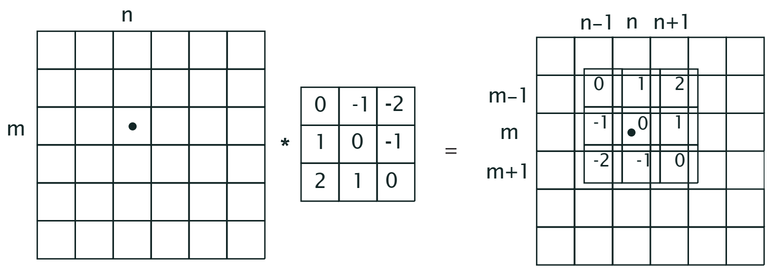

Figure 4.1: *Illustration of the discrete convolution operation with a* 3×3 *filter mask.*

the image. We will learn the reason for this reflection in Section 4.2.5. If we want to calculate the result of the convolution at the point $[m, n]^T$, we center the reflected mask at this point, perform the convolution, and write the result back to position $[m, n]^T$ (Fig. 4.1). This operation is performed for all pixels of the image.

Close to the border of the image, when the filter mask extends over the edge of the image, we run into difficulties as we are missing some image points. The theoretically correct way to solve this problem according to the periodicity property discussed in Section 2.3.4, especially equation Eq. (2.42), is to take into account that finite image matrices must be thought of as being repeated periodically. Consequently, when we arrive at the left border of the image, we take the missing points from the right edge of the image. We speak of a *cyclic convolution*. Only this type of convolution will reduce to a multiplication in the Fourier space (Section 2.3).

In practice, this approach is seldom chosen because the periodic repetition is artificial, inherently related to the sampling of the image data in Fourier space. Instead, we add a border area to the image with half the width of the filter mask. Into this border area we write zeros or we extrapolate in one way or another the gray values from the gray values at the edge of the image. The simplest type of extrapolation is to write the gray values of the edge pixels into the border area. Although this approach gives less visual distortion at the edge of the image than cyclic convolution, we do introduce errors at the edge of the image in a border area with a width of half the size of the filter mask. If we choose any type of extrapolation method, the edge pixels receive too much weight. If we set the border area to zero, we introduce horizontal and vertical edges at the image border.

In conclusion, no perfect method exists to handle pixels close to edges correctly with neighborhood operations. In one way or another, errors are introduced. The only safe way to avoid errors is to ensure that

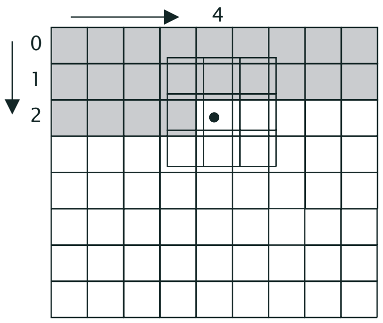

Figure 4.2: *Image convolution by scanning the convolution mask line by line over the image. At the shaded pixels the gray value has already been replaced by the convolution sum. Thus the gray values at the shaded pixels falling within the filter mask need to be stored in an extra buffer.*

objects of interest keep a safe distance from the edge of at least half the size of the largest mask used to process the image.

Equation (4.5) indicates that none of the calculated gray values G'_{mn} will flow into the computation at other neighboring pixels. Thus, if we want to perform the filter operation in-place, we run into a problem. Let us assume that we perform the convolution line by line and from left to right. Then the gray values at all pixel positions above and to the left of the current pixel are already overwritten by the previously computed results (Fig. 4.2).

Consequently, we need to store the gray values at these positions in an appropriate buffer. Efficient algorithms for performing this task are described in Jähne [89] and Jähne et al. [94, Vol. 2, Chap. 5].

The number of elements contained in the mask increases considerably with its size and dimension. A W-dimensional mask with a linear size of R contains R^W elements. The higher the dimension, the faster the number of elements increases with the size of the mask. In higher dimensions, even small neighborhoods include hundreds or thousands of elements.

The challenge for efficient computation schemes is to decrease the number of computations from $O(R^W)$ to a lower order. This means that the number of computations is no longer proportional to R^W but rather to a lower power of R. The ultimate goal is to achieve computation schemes that increase only linearly with the size of the mask ($O(R^1)$) or that do not depend at all on the size of the mask ($O(R^0)$).

4.2.4 Linearity and Shift Invariance

Linear operators are defined by the *principle of superposition*.

Definition 4.3 (Superposition principle) *If $\boldsymbol{G}$ and $\boldsymbol{G}'$ are two W-dimensional complex-valued signals, a and b are two complex-valued scalars, and $\mathcal{H}$ is an operator, then the operator is linear if and only if*

$$\mathcal{H}(a\boldsymbol{G} + b\boldsymbol{G}') = a\mathcal{H}\boldsymbol{G} + b\mathcal{H}\boldsymbol{G}'. \tag{4.15}$$

We can generalize Def. 4.3 to the superposition of many inputs:

$$\mathcal{H}\left(\sum_k a_k \boldsymbol{G}_k\right) = \sum_k a_k \mathcal{H}\boldsymbol{G}_k. \tag{4.16}$$

The superposition states that we can decompose a complex signal into simpler components. We can apply a linear operator to these components and then compose the resulting response from that of the components.

Another important property of an operator is *shift invariance* (also known as *translation invariance* or *homogeneity*). It means that the response of the operator does not depend explicitly on the position in the image. If we shift an image, the output image is the same but for the shift applied. We can formulate this property more elegantly if we define a *shift operator* ${}^{mn}\mathcal{S}$ as

$${}^{mn}\mathcal{S} g_{m'n'} = g_{m'-m,n'-n}. \tag{4.17}$$

Then we can define a *shift-invariant* operator in the following way:

Definition 4.4 (Shift invariance) *An operator is shift invariant if and only if it commutes with the shift operator $\mathcal{S}$:*

$$\mathcal{H}\,{}^{mn}\mathcal{S} = {}^{mn}\mathcal{S}\mathcal{H}. \tag{4.18}$$

From the definition of the convolution operation Eqs. (4.5) and (4.9), it is obvious that it is both linear and shift invariant. This class of operators is called *linear shift-invariant operators* (*LSI operators*). In the context of *time series*, the same property is known as *linear time-invariant* (*LTI*). Note that the shift operator ${}^{mn}\mathcal{S}$ itself is an LSI operator.

4.2.5 Point Spread Function

The linearity and shift-invariance make it easy to understand the response to a convolution operator. As discussed in Section 2.3.1, we can decompose any discrete image (signal) into its individual points or *basis images* ${}^{mn}\boldsymbol{P}$ (Eq. (2.10)):

$$\boldsymbol{G} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} G_{mn}\,{}^{mn}\boldsymbol{P}. \tag{4.19}$$

Linearity says that we can apply an operator to each basis image and then add up the resulting images. Shift invariance says that the response to each of the point images is the same except for a shift. Thus, if we know the response to a point image, we can compute the response to any image.

Consequently, the response to a point image has a special meaning. It is known as the *point spread function* (*PSF*, for time series often denoted as *impulse response*, the response to an impulse). The PSF of a convolution or LSI operator is identical to its mask:

$$p'_{mn} = \sum_{m'=-r}^{r} \sum_{n'=-r}^{r} h_{-m',-n'} \, {}^{00}p_{m+m',n+n'} = h_{m,n} \qquad (4.20)$$

and completely describes a convolution operator in the spatial domain.

The PSF offers another but equivalent view of convolution. The convolution sum in Eq. (4.5) says that each pixel becomes a linear combination of neighboring pixels. The PSF says that each pixel is spread out into the neighborhood as given by the PSF.

4.2.6 Transfer Function

In Section 2.3, we discussed that an image can also be represented in the Fourier domain. This representation is of special importance for linear filters since the convolution operation reduces to a multiplication in the Fourier domain according to the *convolution theorem* (Theorem 2.4, p. 54).

$$\boldsymbol{g} * \boldsymbol{h} \quad \circ\!\!-\!\!\bullet \quad N\hat{\boldsymbol{g}}\hat{\boldsymbol{h}}, \quad \boldsymbol{G} * \boldsymbol{H} \quad \circ\!\!-\!\!\bullet \quad MN\hat{\boldsymbol{G}}\hat{\boldsymbol{H}} \qquad (4.21)$$

The factors N und MN result from the definition of the discrete Fourier transform after Eq. (2.69)b. Therefore we include the factors N and MN, respectively, into the definition of the transfer function. This means that in all further equations $N\hat{\boldsymbol{h}}$ and $MN\hat{\boldsymbol{H}}$ is replaced by $\hat{\boldsymbol{h}}$ and $\hat{\boldsymbol{H}}$, respectively.

The Fourier transform of the convolution mask or PSF is known as the *transfer function* (*TF*) of the linear filter. The transfer function has an important practical meaning. For each wave number, it gives the factor by which a periodic structure is multiplied using the filter operation.

Note that this factor is a *complex number* (Section 2.3.1). Thus a periodic structure experiences not only a change in the amplitude but also a phase shift:

$$\begin{aligned} \hat{g}'_{u,v} = \hat{h}_{u,v}\hat{g}_{u,v} &= r_h \exp(\mathrm{i}\varphi_h)\, r_g \exp(\mathrm{i}\varphi_g) \\ &= r_h r_g \exp[\mathrm{i}(\varphi_h + \varphi_g)], \end{aligned} \qquad (4.22)$$

where the complex numbers are represented in the second part of the equation with their magnitude and phase as complex exponentials.

The symmetry of the filter masks, as discussed in Section 4.2.2, simplifies the transfer function considerably. We can then combine the corresponding symmetric terms in the Fourier transform of the PSF:

$$\begin{aligned}\hat{h}_v &= \sum_{n'=-R}^{R} h_{n'} \exp\left(-\frac{2\pi i n v}{N}\right) \quad (\text{with} \quad h_{-n'} = \pm h_{n'}) \\ &= h_0 + \sum_{n'=1}^{R} h_{n'} \left(\exp\left(-\frac{2\pi i n v}{N}\right) \pm \exp\left(\frac{2\pi i n v}{N}\right)\right).\end{aligned} \tag{4.23}$$

These equations can be further simplified by replacing the discrete wave number by the scaled continuous wave number

$$\tilde{k} = 2v/N, \quad \text{with} \quad -N/2 \le v < N/2. \tag{4.24}$$

The scaled wave number $\tilde{k}$ is confined to the interval $[-1, 1[$. A wave number at the edge of this interval corresponds to the maximal wave number that meets the sampling theorem (Section 9.2.3).

Using the Euler equation $\exp(ix) = \cos x + i \sin x$, Eq. (4.23) reduces for 1-D even and odd filters to:

$$\boxed{\begin{aligned}\text{even:} \quad \hat{h}(\tilde{k}) &= h_0 + 2\sum_{n'=1}^{R} h_{n'} \cos(n'\pi\tilde{k}) \\ \text{odd:} \quad \hat{h}(\tilde{k}) &= -2i \sum_{n'=1}^{R} h_{n'} \sin(n'\pi\tilde{k}).\end{aligned}} \tag{4.25}$$

Correspondingly, a $(2R + 1) \times (2R + 1)$ mask with even horizontal and vertical symmetry results in the transfer function

$$\begin{aligned}\hat{h}(\tilde{\boldsymbol{k}}) &= h_{00} \\ &+ 2\sum_{n'=1}^{R} h_{0n'} \cos(n'\pi\tilde{k}_1) + 2\sum_{m'=1}^{R} h_{m'0} \cos(m'\pi\tilde{k}_2) \\ &+ 4\sum_{m'=1}^{R}\sum_{n'=1}^{R} h_{m'n'} \cos(n'\pi\tilde{k}_1)\cos(m'\pi\tilde{k}_2).\end{aligned} \tag{4.26}$$

Similar equations are valid for other symmetry combinations.

Equations (4.25) and (4.26) are very useful, because they give a straightforward relationship between the coefficients of a *filter mask* and the *transfer function*. They will be our main tool to study the properties of filters for specific image processing tasks in Chapters 11–15.

4.2.7 Further Properties

In this section, we discuss some further properties of convolution operators that will be useful for image and signal processing.

Property 4.1 (Commutativity) *LSI operators are commutative:*

$$\mathcal{H}\mathcal{H}' = \mathcal{H}'\mathcal{H}, \tag{4.27}$$

i. e., the order in which we apply convolution operators to an image does not matter. This property is easy to prove in the Fourier domain, because there the operators reduce to a commutative multiplication.

Property 4.2 (Associativity) *LSI operators are associative:*

$$\mathcal{H}'\mathcal{H}'' = \mathcal{H}. \tag{4.28}$$

Because LSI operations are associative, we can compose a complex operator out of simple operators. Likewise, we can try to decompose a given complex operator into simpler operators. This feature is essential for an effective implementation of convolution operators. As an example, we consider the operator

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}. \tag{4.29}$$

We need 25 multiplications and 24 additions per pixel with this convolution mask. We can easily verify, however, that we can decompose this mask into a horizontal and vertical mask:

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = [1\ 4\ 6\ 4\ 1] * \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix}. \tag{4.30}$$

Applying the two convolutions with the smaller masks one after the other, we need only 10 multiplications and 8 additions per pixel when the operation is applied to the entire image. Filter masks which can be decomposed into one-dimensional masks along the axes are called *separable masks*. We will denote one-dimensional operators with an index indicating the axis. We can then write a separable operator $\mathcal{B}$ in a three-dimensional space:

$$\mathcal{B} = \mathcal{B}_z\mathcal{B}_y\mathcal{B}_x. \tag{4.31}$$

In case of one-dimensional masks directed in orthogonal directions, the convolution reduces to an outer product. Separable filters are more efficient the higher the dimension of the space. Let us consider a $9 \times 9 \times 9$

filter mask as an example. A direct implementation would cost 729 multiplications and 728 additions per pixel, while a separable mask of the same size would need just 27 multiplications and 24 additions, a factor of about 30 fewer operations.

Property 4.3 (Distributivity over Addition) *LSI operators are distributive over addition:*

$$\mathcal{H}' + \mathcal{H}'' = \mathcal{H}. \tag{4.32}$$

Because LSI operators are elements of the same vector space to which they are applied, we can define addition of the operators by the addition of the vector elements. Because of this property we can also integrate operator additions and subtractions into our general operator notation introduced in Section 4.1.4.

4.2.8 Error Propagation with Filtering

Filters are applied to measured data that show noise. Therefore it is important to know how the statistical properties of the filtered data can be inferred from those of the original data. In principle, we solved this question in Section 3.3.3. The *covariance matrix* of the linear combination $\boldsymbol{g}' = \boldsymbol{M}\boldsymbol{g}$ of a random vector $\boldsymbol{g}$ is according to Eq. (3.27) given as

$$\mathrm{cov}(\boldsymbol{g}') = \boldsymbol{M}\,\mathrm{cov}(\boldsymbol{g})\boldsymbol{M}^T. \tag{4.33}$$

Now we need to apply this result to the special case of a convolution. First, we consider only 1-D signals. We assume that the covariance matrix of the signal is homogeneous, i. e., depends only on the distance of the points and not the position itself. Then the variance σ^2 for all elements is equal. Furthermore, the values on the off-diagonals are also equal and the covariance matrix takes the simple form

$$\mathrm{cov}(\boldsymbol{g}) = \begin{bmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \ldots & \ldots \\ \sigma_{-1} & \sigma_0 & \sigma_1 & \sigma_2 & \ldots \\ \sigma_{-2} & \sigma_{-1} & \sigma_0 & \sigma_1 & \ldots \\ \vdots & \sigma_{-2} & \sigma_{-1} & \sigma_0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{4.34}$$

where the index indicates the distance between the points and $\sigma_0 = \sigma^2$. Generally, the covariance decreases with increasing pixel distance. Often, only a limited number of covariances σ_p differ from zero. With statistically uncorrelated pixels, only $\sigma_0 = \sigma^2$ is nonzero.

Because the linear combinations described by $\boldsymbol{M}$ have the special form of a convolution, the matrix has the same form as the homogeneous covariance matrix. For a filter with three coefficients $\boldsymbol{M}$ reduces to

$$\boldsymbol{M} = \begin{bmatrix} h_0 & h_{-1} & 0 & 0 & 0 & \ldots \\ h_1 & h_0 & h_{-1} & 0 & 0 & \ldots \\ 0 & h_1 & h_0 & h_{-1} & 0 & \ldots \\ 0 & 0 & h_1 & h_0 & h_{-1} & \ldots \\ 0 & 0 & 0 & h_1 & h_0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \qquad (4.35)$$

Apart from edge effects, the matrix multiplications in Eq. (4.33) reduce to convolution operations. We introduce the autocovariance vector $\boldsymbol{\sigma} = [\ldots, \sigma_{-1}, \sigma_0, \sigma_1, \ldots]^T$. Then we can write Eq. (4.33) as

$$\boldsymbol{\sigma}' = {}^-\boldsymbol{h} * \boldsymbol{\sigma} * \boldsymbol{h} = \boldsymbol{\sigma} * {}^-\boldsymbol{h} * \boldsymbol{h} = \boldsymbol{\sigma} \star (\boldsymbol{h} \star \boldsymbol{h}), \qquad (4.36)$$

where ${}^-\boldsymbol{h}$ is the reflected convolution mask: ${}^-h_n = h_{-n}$. In the last step, we replaced the convolution by a *correlation*. The convolution of $\boldsymbol{\sigma}$ with $\boldsymbol{h} \star \boldsymbol{h}$ can be replaced by a correlation, because the autocorrelation function of a real-valued function is a function of even symmetry.

In the case of uncorrelated data, the autocovariance vector is a delta function and the autocovariance vector of the noise of the filtered vector reduces to

$$\boldsymbol{\sigma}' = \sigma^2(\boldsymbol{h} \star \boldsymbol{h}). \qquad (4.37)$$

For a filter with R coefficients, now $2R - 1$ values of the autocovariance vector are non-zero. This means that in the filtered signal pixels with a maximal distance of $R - 1$ are now correlated with each other.

Because the covariance vector of a convoluted signal can be described by a correlation, we can also compute the change in the *noise spectrum*, i. e., the *power spectrum* of the noise, caused by a convolution operation. It is just required to Fourier transform Eq. (4.36) under consideration of the correlation theorem (≻ R7). Then we get

$$\boxed{\boldsymbol{\sigma}' = \boldsymbol{\sigma} \star (\boldsymbol{h} \star \boldsymbol{h}) \quad \circ\!\!-\!\!\!-\!\!\bullet \quad \hat{\boldsymbol{\sigma}}'(k) = \hat{\boldsymbol{\sigma}}(k) \left|\hat{\boldsymbol{h}}(k)\right|^2.} \qquad (4.38)$$

This means that the noise spectrum of a convolved signal is given by the multiplication of the noise spectrum of the input data by the square of the transfer function of the filter. With Eqs. (4.36) and (4.38) we have everything at hand to compute the changes of the statistical parameters of a signal (variance, autocovariance matrix, and noise spectrum) caused by a filter operation. Going back from Eq. (4.38), we can conclude that Eq. (4.36) is not only valid for 1-D signals but for signals with arbitrary dimensions.

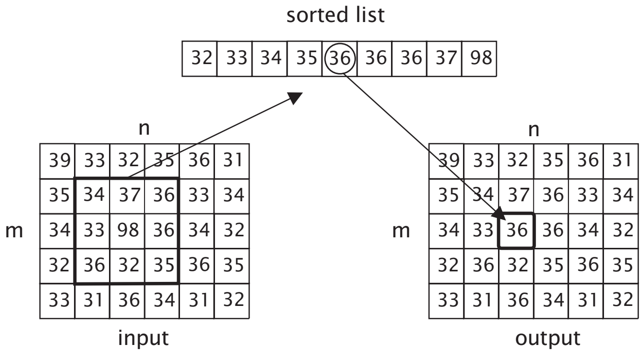

Figure 4.3: *Illustration of the principle of rank value filters with a* 3×3 *median filter.*

4.3 Rank Value Filters

The considerations on how to combine pixels have resulted in the powerful concept of linear shift-invariant systems. Thus we might be tempted to think that we have learnt all we need to know for this type of image processing operation. This is not the case. There is another class of operations which works on a quite different principle.

We might characterize a convolution with a filter mask by weighting and summing up. Comparing and selecting characterize the class of operations to combine neighboring pixels we are considering now. Such a filter is called a *rank-value filter*. For this we take all the gray values of the pixels that lie within the filter mask and sort them by ascending gray value. This sorting is common to all rank value filters. They only differ by the position in the list from which the gray value is picked out and written back to the center pixel. The filter operation which selects the medium value is called the *median filter*. Figure 4.3 illustrates how the median filter works. The filters choosing the minimum and maximum values are denoted as the *minimum* and *maximum filter*, respectively.

The median filter is a nonlinear operator. For the sake of simplicity, we consider a one-dimensional case with a 3-element median filter. It is easy to find two vectors for which the median filter is not linear. First we apply the median filter to the sum of two signals. This results in

$$\mathcal{M}([\cdots\ 0\ 1\ 0\ 0\ \cdots] + [\cdots\ 0\ 0\ 1\ 0\ \cdots]) = [\cdots\ 0\ 1\ 1\ 0\ \cdots].$$

Then we apply the median filter first to the two components before we add the two results:

$$\mathcal{M}[\cdots\ 0\ 1\ 0\ 0\ \cdots] + \mathcal{M}[\cdots\ 0\ 0\ 1\ 0\ \cdots] = [\cdots\ 0\ 0\ 0\ 0\ \cdots].$$

The results of both computations are different. This proves that the median filter is nonlinear.

There are a number of significant differences between convolution filters and rank value filters. Most important, rank value filters belong to the class of *nonlinear filters*. Consequently, it is much more difficult to understand their general properties. As rank value filters do not perform arithmetic operations but select pixels, we will never run into rounding problems. These filters map a discrete set of gray values onto themselves.

4.4 LSI-Filters: Further Properties

4.4.1 Convolution, Linearity, and Shift Invariance

In Section 4.2.4 we saw that a convolution operator is a linear shift invariant operator. But is the reverse also true that *any* linear shift-invariant operator is also a convolution operator? In this section we are going to prove this statement. From our considerations in Section 4.2.5, we are already familiar with the *point spread function* of continuous and discrete operators. Here we introduce the formal definition of the point spread function for an operator $\mathcal{H}$ onto an $M \times N$-dimensional vector space:

$$\boldsymbol{H} = \mathcal{H}\,{}^{00}\boldsymbol{P}. \tag{4.39}$$

Now we can use the linearity Eq. (4.16) and the shift invariance Eq. (4.18) of the operator $\mathcal{H}$ and the definition of the impulse response Eq. (4.39) to calculate the result of the operator on an arbitrary image $\boldsymbol{G}$ in the space domain

$$\begin{array}{rcll}
(\mathcal{H}\boldsymbol{G})_{mn} & = & \left[\mathcal{H}\left[\sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}\,{}^{m'n'}\boldsymbol{P}\right]\right]_{mn} & \text{with Eq. (4.16)} \\
 & = & \left[\sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}\mathcal{H}\,{}^{m'n'}\boldsymbol{P}\right]_{mn} & \text{linearity} \\
 & = & \left[\sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}\mathcal{H}\,{}^{m'n'}\mathcal{S}\,{}^{00}\boldsymbol{P}\right]_{mn} & \text{with Eq. (4.17)} \\
 & = & \left[\sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}\,{}^{m'n'}\mathcal{S}\mathcal{H}\,{}^{00}\boldsymbol{P}\right]_{mn} & \\
 & = & \left[\sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}\,{}^{m'n'}\mathcal{S}\boldsymbol{H}\right]_{mn} & \text{with Eq. (4.39)} \\
 & = & \sum\limits_{m'=0}^{M-1}\sum\limits_{n'=0}^{N-1} g_{m'n'}h_{m-m',n-n'} & \text{with Eq. (4.17)} \\
 & = & \sum\limits_{m''=0}^{M-1}\sum\limits_{n''=0}^{N-1} g_{m-m'',n-n''}h_{m'',n''} & \begin{array}{l} m'' = m - m' \\ n'' = n - n' \end{array}.
\end{array}$$

These calculations prove that a linear shift-invariant operator must necessarily be a convolution operation in the space domain. There is no other operator type which is both linear and shift invariant.

4.4.2 *Inverse Operators*

Can we invert a filter operation so that we can get back the original image from a filtered image? This question is significant because degradations such as image blurring by motion or by defocused optics can also be regarded as filter operations (Section 7.6.1). If an inverse operator exists and if we know the point spread function of the degradation, we can reconstruct the original, undisturbed image. The problem of inverting a filter operation is known as *deconvolution* or *inverse filtering*.

By considering the filter operation in the Fourier domain, we immediately recognize that we can only reconstruct those wave numbers for which the transfer function of the filter does not vanish. In practice, the condition for inversion of a filter operation is much more restricted because of the limited quality of the image signals. If a wave number is attenuated below a critical level, which depends on the noise and quantization (Section 9.5), it will not be recoverable. It is obvious that these conditions limit the power of a straightforward inverse filtering considerably. The problem of inverse filtering is considered further in Chapter 17.5.

4.4.3 *Eigenfunctions*

Next we are interested in the question whether special types of images $\boldsymbol{E}$ exist which are preserved by a linear shift-invariant operator, except for multiplication with a scalar. Intuitively, it is clear that these images have a special importance for LSI operators. Mathematically speaking, this means

$$\mathcal{H}\boldsymbol{E} = \lambda \boldsymbol{E}. \tag{4.40}$$

A vector (image) which meets this condition is called an *eigenvector* (*eigenimage*) or *characteristic vector* of the operator, the scaling factor λ an *eigenvalue* or *characteristic value* of the operator.

In order to find the eigenimages of LSI operators, we discuss the shift operator $\mathcal{S}$. It is quite obvious that for real images only a trivial eigenimage exists, namely a constant image. For complex images, however, a whole set of eigenimages exists. We can find it when we consider the shift property of the *complex exponential*

$$^{uv}w_{mn} = \exp\left(\frac{2\pi\mathrm{i}mu}{M}\right)\exp\left(\frac{2\pi\mathrm{i}nv}{N}\right), \tag{4.41}$$

which is given by

$$^{kl}\mathcal{S}\,^{uv}\boldsymbol{W} = \exp\left(-\frac{2\pi\mathrm{i}ku}{M}\right)\exp\left(-\frac{2\pi\mathrm{i}lv}{N}\right)\,^{uv}\boldsymbol{W}. \tag{4.42}$$

The latter equation directly states that the complex exponentials $^{uv}\boldsymbol{W}$ are eigenfunctions of the shift operator. The eigenvalues are complex phase factors which depend on the wave number indices (u, v) and the shift (k, l). When the shift is one wavelength, $(k, l) = (M/u, N/v)$, the phase factor reduces to 1 as we would expect.

Now we are curious to learn whether any linear shift-invariant operator has such a handy set of eigenimages. It turns out that all linear shift-invariant operators have the same set of eigenimages. We can prove this statement by referring to the *convolution theorem* (Section 2.3, Theorem 2.4, p. 54) which states that convolution is a point-wise multiplication in the Fourier space. Thus each element of the image representation in the Fourier space $\hat{g}_{uv}$ is multiplied by the complex scalar $\hat{h}_{uv}$. Each point in the Fourier space represents a base image, namely the complex exponential ${}^{uv}\boldsymbol{W}$ in Eq. (4.41) multiplied with the scalar $\hat{g}_{uv}$. Therefore, the complex exponentials are eigenfunctions of any convolution operator. The eigenvalues are then the elements of the transfer function, $\hat{h}_{uv}$. In conclusion, we can write

$$\mathcal{H}(\hat{g}_{uv}\,{}^{uv}\boldsymbol{W}) = \hat{h}_{uv}\hat{g}_{uv}\,{}^{uv}\boldsymbol{W}. \tag{4.43}$$

The fact that the eigenfunctions of LSI operators are the basis functions of the Fourier domain explains why convolution reduces to a multiplication in Fourier space and underlines the central importance of the Fourier transform for image processing.

4.5 *Recursive Filters*

4.5.1 *Introduction*

As convolution requires many operations, the question arises whether it is possible or even advantageous to include the already convolved neighboring gray values into the convolution at the next pixel. In this way, we might be able to do a convolution with fewer operations. In effect, we are able to perform convolutions with much less computational effort and also more flexibility. However, these filters, which are called *recursive filters*, are much more difficult to understand and to handle — especially in the multidimensional case.

For a first impression, we consider a very simple example. The simplest 1-D recursive filter we can think of has the general form

$$g'_n = \alpha g'_{n-1} + (1-\alpha)g_n. \tag{4.44}$$

This filter takes the fraction $1-\alpha$ from the previously calculated value and the fraction α from the current pixel. Recursive filters, in contrast to nonrecursive filters, work in a certain direction, in our example from left to right. For time series, the preferred direction seems natural, as the current state of a signal depends only on previous values. Filters that depend only on the previous values of the signal are called *causal filters*. For spatial data, however, no preferred direction exists. Consequently, we have to search for ways to construct filters with even and odd symmetry as they are required for image processing from recursive filters.

With recursive filters, the point spread function is no longer identical to the filter mask, but must be computed. From Eq. (4.44), we can calculate the *point spread function* or *impulse response* of the filter as the response of the filter to the *discrete delta function* (Section 4.2.5)

$$\delta_n = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}. \tag{4.45}$$

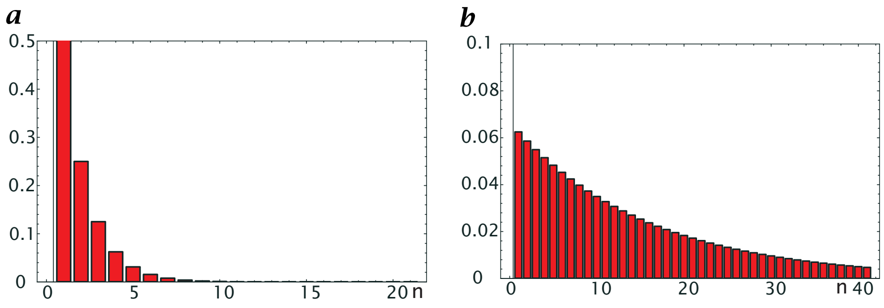

Figure 4.4:** Point spread function of the recursive filter* $g'_n = \alpha g'_{n-1} + (1-\alpha)g_n$ *for **a $\alpha = 1/2$ *and **b*** $\alpha = 15/16$.

Recursively applying Eq. (4.44), we obtain

$$g'_{-1} = 0, \quad g'_0 = 1-\alpha, \quad g'_1 = (1-\alpha)\alpha, \quad \ldots, \quad g'_n = (1-\alpha)\alpha^n. \tag{4.46}$$

This equation shows three typical general properties of recursive filters:

- First, the impulse response is infinite (Fig. 4.4), despite the finite number of coefficients. For $|\alpha| < 1$ it decreases exponentially but never becomes exactly zero. In contrast, the impulse response of nonrecursive convolution filters is always finite. It is equal to the size of the filter mask. Therefore the two types of filters are sometimes named *finite impulse response filters* (*FIR filter*) and *infinite impulse response filters* (*IIR filter*).
- FIR filters are always *stable*. This means that the impulse response is finite. Then the response of a filter to any finite signal is finite. This is not the case for IIR filters. The stability of recursive filters depends on the filter coefficients. The filter in Eq. (4.44) is unstable for $|\alpha| > 1$, because then the impulse response diverges. In the simple case of Eq. (4.44) it is easy to recognize the instability of the filter. Generally, however, it is much more difficult to analyze the stability of a recursive filter, especially in two dimensions and higher.
- Any recursive filter can be replaced by a nonrecursive filter, in general with an infinite-sized mask. Its mask is given by the point spread function of the recursive filter. The inverse conclusion does not hold. This can be seen by the very fact that a non-recursive filter is always stable.

4.5.2 *Transfer Function, z-Transform, and Stable Response*

After this introductory example, we are ready for a more formal discussion of *recursive filters*. Recursive filters include results from previous convolutions at neighboring pixels into the convolution sum and thus become directional. We discuss here only 1-D recursive filters. The general equation for a filter running from left to right is

$$\boxed{g'_n = -\sum_{n''=1}^{S} a_{n''} g'_{n-n''} + \sum_{n'=-R}^{R} h_{n'} g_{n-n'}.} \tag{4.47}$$

While the neighborhood of the nonrecursive part (coefficients h) is symmetric around the central point, the recursive part (coefficients a) uses only previously computed values. Such a recursive filter is called a *causal filter*.

If we put the recursive part on the left hand side of the equation, we observe that the recursive filter is equivalent to the following difference equation, also known as an *ARMA*(S,R) process (*autoregressive moving average process*):

$$\sum_{n''=0}^{S} a_{n''} g'_{n-n''} = \sum_{n'=-R}^{R} h_{n'} g_{n-n'} \quad \text{with} \quad a_0 = 1. \tag{4.48}$$

The transfer function of such a filter with a recursive and a nonrecursive part can be computed by applying the *discrete Fourier transform* (Section 2.3.2) and making use of the shift theorem (Theorem 2.3, p. 54). Then

$$\hat{g}'(k) \sum_{n''=0}^{S} a_{n''} \exp(-2\pi \mathrm{i} n'' k) = \hat{g}(k) \sum_{n'=-R}^{R} h_{n'} \exp(-2\pi \mathrm{i} n' k). \tag{4.49}$$

Thus the *transfer function* is

$$\boxed{\hat{h}(k) = \frac{\hat{g}'(k)}{\hat{g}(k)} = \frac{\sum\limits_{n'=-R}^{R} h_{n'} \exp(-2\pi \mathrm{i} n' k)}{\sum\limits_{n''=0}^{S} a_{n''} \exp(-2\pi \mathrm{i} n'' k)}.} \tag{4.50}$$

The zeros of the numerator and the denominator govern the properties of the transfer function. Thus, a zero in the nonrecursive part of the transfer function causes a zero in the transfer function, i. e., vanishing of the corresponding wave number. A zero in the recursive part causes a pole in the transfer function, i. e., an infinite response.

A determination of the zeros and thus a deeper analysis of the transfer function is not possible from Eq. (4.50). It requires an extension similar to the extension from real numbers to complex numbers that was used to introduce the Fourier transform (Section 2.3.2). We observe that the expressions for both the numerator and the denominator are polynomials in the *complex exponential* $\exp(2\pi \mathrm{i} k)$ of the form

$$\sum_{n=0}^{S} a_n \left(\exp(-2\pi \mathrm{i} k)\right)^n. \tag{4.51}$$

The complex exponential has a magnitude of one and thus covers the unit circle in the complex plane. The zeros of the polynomial need not to be located one the unit circle but can be an arbitrary complex number. Therefore, it is useful to extend the polynomial so that it covers the whole complex plane. This is possible with the expression $z = r\exp(2\pi \mathrm{i} k)$ that describes a circle with the radius r in the complex plane.

With this extension we obtain a polynomial of the complex number z. As such we can apply the fundamental law of algebra that states that any polynomial of degree N can be factorized into N factors containing the roots or zeros of the polynomial:

$$\sum_{n=0}^{N} a_n z^n = a_N z^N \prod_{n=1}^{N} \left(1 - r_n z^{-1}\right). \tag{4.52}$$

With Eq. (4.52) we can factorize the recursive and nonrecursive parts of the polynomials in the transfer function into the following products:

$$\begin{array}{rcccl} \sum\limits_{n=0}^{S} a_n z^{-n} & = & z^{-S} \sum\limits_{n'=0}^{S} a_{S-n'} z^{n'} & = & \prod\limits_{n=1}^{S} (1 - d_n z^{-1}), \\ \sum\limits_{n=-R}^{R} h_n z^{-n} & = & z^{-R} \sum\limits_{n'=0}^{2R} h_{R-n'} z^{n'} & = & h_{-R} z^R \prod\limits_{n=1}^{2R} (1 - c_n z^{-1}). \end{array} \tag{4.53}$$

With $z = \exp(2\pi i k)$ the transfer function can finally be written as

$$\boxed{\hat{h}(z) = h_{-R} z^R \frac{\prod\limits_{n'=1}^{2R} (1 - c_{n'} z^{-1})}{\prod\limits_{n''=1}^{S} (1 - d_{n''} z^{-1})}.} \tag{4.54}$$

Each of the factors $c_{n'}$ and $d_{n''}$ is a zero of the corresponding polynomial ($z = c_{n'}$ or $z = d_{n''}$).

The inclusion of the factor r in the extended transfer function results in an extension of the Fourier transform, the *z-transform*, which is defined as

$$\hat{g}(z) = \sum_{n=-\infty}^{\infty} g_n z^{-n}. \tag{4.55}$$

The z-transform of the series g_n can be regarded as the Fourier transform of the series $g_n r^{-n}$ [124]. The z-transform is the key mathematical tool to understand 1-D recursive filters. It is the discrete analogue to the *Laplace transform*. Detailed accounts of the z-transform are given by Oppenheim and Schafer [148] and Poularikas [156]; the 2-D z-transform is discussed by Lim [124].

Now we analyze the transfer function in more detail. The factorization of the transfer function is a significant advantage because each factor can be regarded as an individual filter. Thus each recursive filter can be decomposed into a cascade of simple recursive filters. As the factors are all of the form

$$f_n(\tilde{k}) = 1 - d_n \exp(-2\pi i \tilde{k}) \tag{4.56}$$

and the impulse response of the filter must be real, the transfer function must be Hermitian, that is, $f(-k) = f^*(k)$. This can only be the case when either the zero d_n is real or a pair of factors exists with complex-conjugate zeros. This condition gives rise to two basic types of recursive filters, the *relaxation filter* and the *resonance filter* that are discussed in detail in Sections 4.5.5 and 4.5.6.

4.5.3 *Higher-Dimensional Recursive Filters*

Recursive filters can also be defined in higher dimensions with the same type of equation as in Eq. (4.47); also the transfer function and z-transform of higher-dimensional recursive filters can be written in the very same way as in Eq. (4.50). However, it is generally not possible to factorize the z-transform as in Eq. (4.54) [124]. From Eq. (4.54) we can immediately conclude that it will be possible to factorize a *separable* recursive filter because then the higher-dimensional polynomials can be factorized into 1-D polynomials. Given these inherent mathematical difficulties of higher-dimensional recursive filters, we will restrict the further discussion on 1-D recursive filters.

4.5.4 *Symmetric Recursive Filtering*

While a filter that uses only previous data is natural and useful for real-time processing of time series, it makes little sense for spatial data. There is no "before" and "after" in spatial data. Even worse is the signal-dependent spatial shift (delay) associated with recursive filters.

With a single recursive filter it is impossible to construct a so-called *zero-phase filter* with an even transfer function. Thus it is necessary to combine multiple recursive filters. The combination should either result in a zero-phase filter suitable for smoothing operations or a derivative filter that shifts the phase by 90°. Thus the transfer function should either be purely real or purely imaginary (Section 2.3.4).

We start with a 1-D causal recursive filter that has the transfer function

$$^{+}\hat{h}(\tilde{k}) = a(\tilde{k}) + \mathrm{i}b(\tilde{k}). \tag{4.57}$$

The superscript "+" denotes that the filter runs in positive coordinate direction. The transfer function of the same filter but running in the opposite direction has a similar transfer function. We replace $\tilde{k}$ by $-\tilde{k}$ and note that $a(-\tilde{k}) = a(+\tilde{k})$ and $b(-\tilde{k}) = -b(\tilde{k}))$, because the transfer function of a real PSF is Hermitian (Section 2.3.4), and obtain

$$^{-}\hat{h}(\tilde{k}) = a(\tilde{k}) - \mathrm{i}b(\tilde{k}). \tag{4.58}$$

Thus, only the sign of the imaginary part of the transfer function changes when the filter direction is reversed.

We now have three possibilities to combine the two transfer functions (Eqs. (4.57) and (4.58)) either into a purely real or imaginary transfer function:

$$\begin{array}{ll} \text{Addition} & {}^{e}\hat{h}(\tilde{k}) = \dfrac{1}{2}\left({}^{+}\hat{h}(\tilde{k}) + {}^{-}\hat{h}(\tilde{k})\right) = a(\tilde{k}), \\ \text{Subtraction} & {}^{o}\hat{h}(\tilde{k}) = \dfrac{1}{2}\left({}^{+}\hat{h}(\tilde{k}) - {}^{-}\hat{h}(\tilde{k})\right) = \mathrm{i}b(\tilde{k}), \\ \text{Multiplication} & \hat{h}(\tilde{k}) = {}^{+}\hat{h}(\tilde{k})\,{}^{-}\hat{h}(\tilde{k}) = a^2(\tilde{k}) + b^2(\tilde{k}). \end{array} \tag{4.59}$$

Addition and multiplication (consecutive application) of the left and right running filter yields filters of even symmetry and a real transfer function, while subtraction results in a filter of odd symmetry and a purely imaginary transfer function.

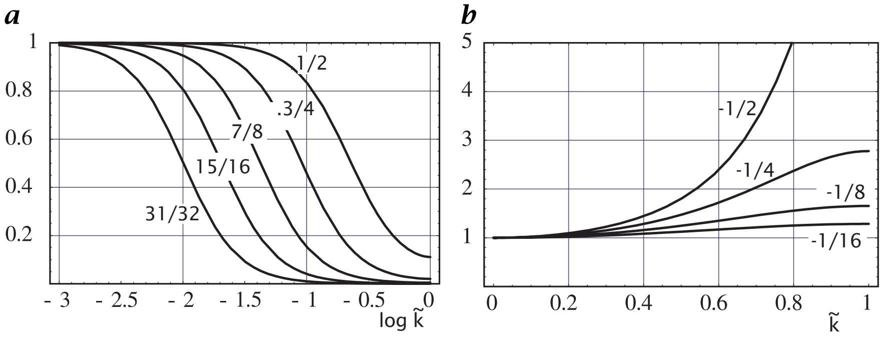

Figure 4.5: *Transfer function of the relaxation filter $g'_n = \alpha g'_{n\mp1} + (1-\alpha)g_n$ applied first in forward and then in backward direction for **a** positive; and **b** negative values of α as indicated.*

4.5.5 *Relaxation Filters*

The simple recursive filter discussed in Section 4.5.1

$$g'_n = a_1 g'_{n\mp1} + h_0 g_n \quad \text{with} \quad a_1 = \alpha,\ h_0 = (1-\alpha) \tag{4.60}$$

and the point spread function

$$^{\pm}r_{\pm n} = \begin{cases} (1-\alpha)\alpha^n & n \geq 0 \\ 0 & \text{else} \end{cases} \tag{4.61}$$

is a *relaxation filter*. The transfer function of the filter running either in forward or in backward direction is, according to Eq. (4.50) with Eq. (4.60), given by

$$^{\pm}\hat{r}(\tilde{k}) = \frac{1-\alpha}{1-\alpha\exp(\mp\pi\mathrm{i}\tilde{k})} \quad \text{with} \quad \alpha \in \mathbb{R}. \tag{4.62}$$

The transfer function Eq. (4.62) is complex and can be divided into its real and imaginary parts as

$$^{\pm}\hat{r}(\tilde{k}) = \frac{1-\alpha}{1-2\alpha\cos\pi\tilde{k}+\alpha^2}\left[(1-\alpha\cos\pi\tilde{k}) \mp \mathrm{i}\alpha\sin\pi\tilde{k}\right]. \tag{4.63}$$

After Eq. (4.59), we can then compute the transfer function $\hat{r}$ for the resulting symmetric filter if we apply the relaxation filters successively in positive and negative direction:

$$\boxed{\hat{r}(\tilde{k}) = {}^{+}\hat{r}(\tilde{k})\,{}^{-}\hat{r}(\tilde{k}) = \frac{(1-\alpha)^2}{1-2\alpha\cos\pi\tilde{k}+\alpha^2} = \frac{1}{(1+\beta)-\beta\cos\pi\tilde{k}}} \tag{4.64}$$

with

$$\beta = \frac{2\alpha}{(1-\alpha)^2} \quad \text{and} \quad \alpha = \frac{1+\beta-\sqrt{1+2\beta}}{\beta}$$

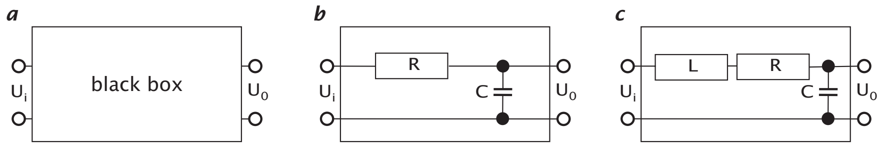

Figure 4.6: *Analog filter for time series.* ***a*** *Black-box model: a signal U_i is put into an unknown system and at the output we measure the signal U_o.* ***b*** *A resistor-capacitor circuit as a simple example of an analog lowpass filter.* ***c*** *Damped resonance filter consisting of an inductor L, a resistor R, and a capacitor C.*

From Eq. (4.61) we can conclude that the relaxation filter is stable if $|\alpha| < 1$, which corresponds to $\beta \in]-1/2, \infty[$. As already noted, the transfer function is one for small wave numbers. A Taylor series in $\tilde{k}$ results in

$$\hat{r}(\tilde{k}) \approx 1 - \frac{\alpha}{(1-\alpha)^2}(\pi\tilde{k})^2 + \frac{\alpha((1+10\alpha+\alpha^2)}{12(1-\alpha^2)^2}(\pi\tilde{k})^4. \tag{4.65}$$

If α is positive, the filter is a low-pass filter (Fig. 4.5a). It can be tuned by adjusting α. If α is approaching 1, the averaging distance becomes infinite. For negative α, the filter enhances high wave numbers (Fig. 4.5b).

This filter is the discrete analog to the first-order differential equation $\dot{y} + \tau y = 0$ describing a relaxation process with the relaxation time $\tau = -\Delta t / \ln \alpha$.

An example is the simple resistor-capacitor circuit shown in Fig. 4.6b. The differential equation for this filter can be derived from Kirchhoff's current-sum law. The current flowing through the resistor from U_i to U_o must be equal to the current flowing into the capacitor. Since the current flowing into a capacitor is proportional to the temporal derivative of the potential U_o, we end up with the first-order differential equation

$$\frac{U_i - U_o}{R} = C\frac{\partial U_o}{\partial t}. \tag{4.66}$$

and the time constant is given by $\tau = RC$.

4.5.6 *Resonance Filters*

The second basic type of a recursive filter that we found from the discussion of the transfer function in Section 4.5.2 has a pair of complex-conjugate zeros. Therefore, the transfer function of this filter running in forward or backward direction is

$$\begin{aligned} {}^{\pm}\hat{s}(\tilde{k}) &= \frac{1}{(1 - r\exp(\mathrm{i}\pi\tilde{k}_0)\exp(\mp\mathrm{i}\pi\tilde{k}))(1 - r\exp(-\mathrm{i}\pi\tilde{k}_0)\exp(\mp\mathrm{i}\pi\tilde{k}))} \\ &= \frac{1}{1 - 2r\cos(\pi\tilde{k}_0)\exp(\mp\mathrm{i}\pi\tilde{k}) + r^2\exp(\mp 2\mathrm{i}\pi\tilde{k})}. \end{aligned} \tag{4.67}$$

The second row of the equation shows that this recursive filter has the coefficients $h_0 = 1$, $a_1 = -2r\cos(\pi\tilde{k}_0)$, and $a_2 = r^2$ so that:

$$g'_n = g_n + 2r\cos(\pi\tilde{k}_0)g'_{n\mp1} - r^2 g'_{n\mp2}. \tag{4.68}$$

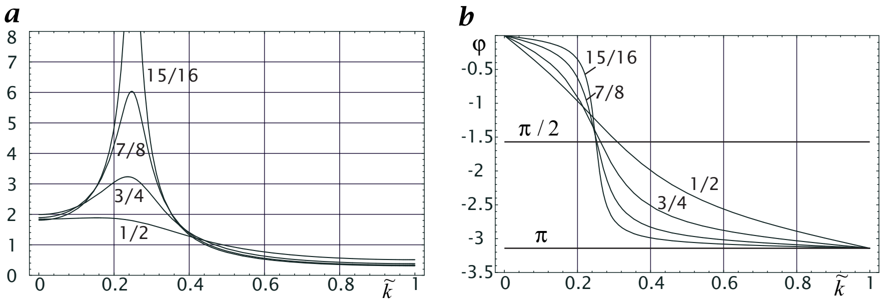

Figure 4.7: ***a*** *Magnitude and* ***b*** *phase shift of the transfer function of the resonance filter according to Eq. (4.67) for* $\tilde{k}_0 = 1/4$ *and values for* r *as indicated.*

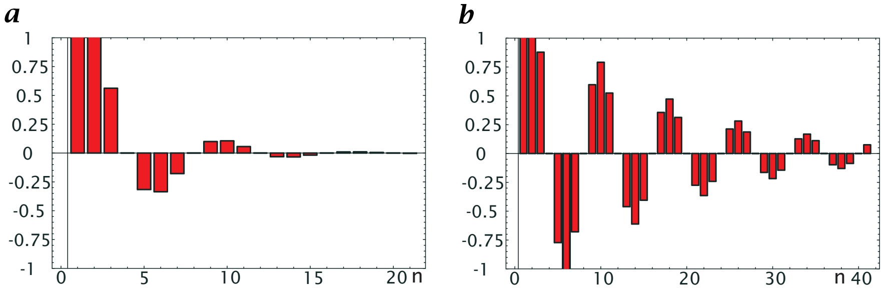

Figure 4.8: *Point spread function of the recursive resonance filter according to Eq. (4.68) for* ***a*** $\tilde{k}_0 = 1/4$, $r = 3/4$ *and* ***b*** $\tilde{k}_0 = 1/4$, $r = 15/16$.

From the transfer function in Eq. (4.67) we conclude that this filter is a *bandpass filter* with a passband wave number of $\pm\tilde{k}_0$ (Fig. 4.7). For $r = 1$ the transfer function has two poles at $\tilde{k} = \pm\tilde{k}_0$.

The impulse response of this filter is after [148]

$$h_{\pm n} = \begin{cases} \dfrac{r^n}{\sin \pi\tilde{k}_0} \sin[(n+1)\pi\tilde{k}_0] & n \geq 0 \\ 0 & n < 0 \end{cases}. \tag{4.69}$$

This means that the filter acts as a damped oscillator. The parameter $\tilde{k}_0$ gives the wave number of the oscillation and the parameter r is the damping constant (Fig. 4.8). The filter is only stable if $r \leq 1$.

If we run the filter back and forth, the resulting filter has a real transfer function $\hat{s}(\tilde{k}) = {}^{+}\hat{s}(\tilde{k})\,{}^{-}\hat{s}(\tilde{k})$ that is given by

$$\hat{s}(\tilde{k}) = \frac{1}{\left(1 - 2r\cos[\pi(\tilde{k}-\tilde{k}_0)] + r^2\right)\left(1 - 2r\cos[\pi(\tilde{k}+\tilde{k}_0)] + r^2\right)}. \tag{4.70}$$

The transfer function of this filter can be normalized so that its maximal value becomes 1 in the passband by setting the nonrecursive filter coefficient h_0 to

$(1-r^2)\sin(\pi\tilde{k}_0)$. Then we obtain the following modified recursion

$$g'_n = (1-r^2)\sin(\pi\tilde{k}_0)g_n + 2r\cos(\pi\tilde{k}_0)g'_{n\mp1} - r^2 g'_{n\mp2}. \tag{4.71}$$

For symmetry reasons, the factors become most simple for a resonance wave number of $\tilde{k}_0 = 1/2$. Then the recursive filter is

$$g'_n = (1-r^2)g_n - r^2 g'_{n\mp2} = g_n - r^2(g_n + g'_{n\mp2}) \tag{4.72}$$

with the transfer function

$$\hat{s}(\tilde{k}) = \frac{(1-r^2)^2}{1+r^4+2r^2\cos(2\pi\tilde{k})}. \tag{4.73}$$

The maximum response of this filter at $\tilde{k} = 1/2$ is one and the minimum response at $\tilde{k} = 0$ and $\tilde{k} = 1$ is $[(1-r^2)/(1+r^2)]^2$.

This resonance filter is the discrete analog to a linear system governed by the second-order differential equation $\ddot{y} + 2\tau\dot{y} + \omega_0^2 y = 0$, the damped harmonic oscillator such as the LRC circuit in Fig. 4.6c. The circular eigenfrequency ω_0 and the time constant τ of a real-world oscillator are related to the parameters of the discrete oscillator, r and $\tilde{k}_0$ by [89]

$$r = \exp(-\Delta t/\tau) \quad \text{and} \quad \tilde{k}_0 = \omega_0\Delta t/\pi. \tag{4.74}$$

4.5.7 *LSI Filters and System Theory*

The last example of the damped oscillator illustrates that there is a close relationship between discrete filter operations and analog physical systems. Thus, digital filters model a real-world physical process. They pattern how the corresponding system would respond to a given input signal g. Actually, we will make use of this equivalence in our discussion of image formation in Chapter 7. There we will find that imaging with a homogeneous optical system is completely described by its point spread function and that the image formation process can be described by convolution. Optical imaging together with physical systems such as electrical filters and oscillators of all kinds can thus be regarded as representing an abstract type of process or system, called a *linear shift-invariant system* or short *LSI*.

This generalization is very useful for image processing, as we can describe both image formation and image processing as convolution operations with the same formalism. Moreover, the images observed may originate from a physical process that can be modeled by a linear shift-invariant system. Then the method for finding out how the system works can be illustrated using the black-box model (Fig. 4.6a). The black box means that we do not know the composition of the system observed or, physically speaking, the laws that govern it. We can find them out by probing the system with certain signals (input signals) and watching the response by measuring some other signals (output signals). If it turns out that the system is linear, it will be described completely by the impulse response.

Many biological and medical experiments are performed in this way. Biological systems are typically so complex that the researchers often stimulate them with signals and watch for responses in order to find out how they work and to construct a model. From this model more detailed research may start to investigate how the observed system functions might be realized. In this way many properties of biological visual systems have been discovered. But be careful — a model is not the reality! It pictures only the aspect that we probed with the applied signals.

4.6 *Exercises*

4.1: General properties of convolution operators

Interactive demonstration of general properties of linear shiftinvariant operators (dip6ex04.01).

4.2: *1-D convolution

Examine the following 1-D convolution masks:

$$\begin{array}{rc}
a) & 1/4[1\ 2\ 1] \\
b) & 1/4[1\ 0\ 2\ 0\ 1] \\
c) & 1/16[1\ 2\ 3\ 4\ 3\ 2\ 1] \\
d) & 1/2[1\ 0\ -1] \\
e) & [1\ -2\ 1] \\
f) & [1\ 0\ -2\ 0\ 1]
\end{array}$$

Answer the following questions:

1. Which symmetry do these convolution masks show?
2. Compute the transfer functions. Try to obtain the simplest possible equation by using trigonometric identities for half and double angles.
3. Check the computed transfer functions by applying the masks to a constant gray value structure ($\tilde{k} = 0$)

$$\ldots \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad \ldots,$$

   a gray value structure with the maximal possible wave number ($\tilde{k} = 1$)

$$\ldots \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad \ldots$$

   and a step edge

$$\ldots \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad \ldots.$$

4.3: **2-D convolution

Answer the same questions as in Exercise 4.2 for the following 2-D convolution masks:

$$a)\ \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad b)\ \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$

$$c)\ \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad d)\ \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

Check if the masks are separable or can be composed in another way from the 1-D convolution masks of Exercise 4.2. This saves you a lot of computational work!

4.4: *Commutativity and associativity of convolution

Show by applying the convolution masks a) and d) from Exercise 4.2 to a step edge

$$\ldots\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ \ldots$$

that convolution is commutative and associative.

4.5: *Convolution masks with even number of coefficients

Also for filters with an even number of coefficients ($2R$), it is possible to define filters with even and odd symmetry if we imagine the convolution result is put on an intermediate grid. The convolution mask can be written as

$$[h_{-R}, \ldots, h_{-1}, h_1, \ldots, h_R].$$

The reference part (≻ R11) gives the equations for the transfer functions of these masks.

1. Prove these equations by applying a shift of half a grid distance to the general equation for the transfer function Eq. (4.23).
2. Compute the transfer functions of the two elementary masks [1 1]/2 (mean of two neighboring points) and [1 − 1] (difference of two neighboring points).

4.6: ** Manipulations of convolution masks

Examine how the transfer function of a convolution mask with $(2R + 1)$-coefficients changes if you change the coefficients in the following way:

1. Complimentary filter

$$h'_n = \delta_n - h_n$$

   Example: [1 1 1]/3 change to [−1 2 − 1]/3
2. Partial sign change

$$h'_n = \begin{cases} h_n & n \text{ even} \\ -h_n & n \text{ odd} \end{cases}$$

   Example: [1 2 1]/4 changes to [−1 2 − 1]/4
3. Streching

$$h'_n = \begin{cases} h_{n/2} & n \text{ even} \\ 0 & n \text{ odd} \end{cases}$$

   Example [1 2 1]/4 changes to [1 0 2 0 1]/4

4.7: *****Inverse convolution**

Does an inverse operator exist for the following convolution operators?

$$\begin{array}{ll} a) & 1/6[1\ 4\ 1] \\ b) & 1/4[1\ 2\ 1] \\ c) & 1/3[1\ 1\ 1] \end{array}$$

Are these inverse operators again a convolution operator? (see Section 4.4.2) If yes, do they have a special structure?

4.8: ****Change of statistics of 1-D signals by convolution**

Compute the autocovariance vector of an uncorrelated time series with constant variance σ^2 for all elements that have been convolved with the filters a), d), and e) from Exercise 4.2. Analyze the results, especially for the variance of the convolved time series.

4.9: Recursive relaxation filters

Interactive demonstration of recursive relaxation filters (dip6ex04.02).

4.10: Recursive resonance filters

Interactive demonstration of recursive resonance filters (dip6ex04.03).

4.11: ****Stability of recursive filters**

1. Which of the following recursive filters (Section 4.5) are stable?

$$\begin{array}{ll} a) & g'_n = -1/4g'_{n-1} + 5/4g_n \\ b) & g'_n = 5/4g'_{n-1} - 1/4g_n \\ c) & g'_n = -1/4g'_{n-2} + 3/4g_n \\ d) & g'_n = -5/4g'_{n-2} - 1/4g_n \end{array}$$

   Answer this question by computing the point spread function.
2. Compute the transfer functions of these filters.

4.12: ****Physical systems and recursive filters**

Physical systems can be regarded as implementations of recursive filters. Compute the point spread function (impulse response) and transfer function of the following physical systems:

1. A cascaded electric lowpass filter consisting of two stages each with a resistor R and a capacity C.
2. A spring pendulum with a mass m, a spring constant D ($K = Dx$) and a friction coefficient k ($K = k\mathrm{d}x/\mathrm{d}t$).

4.13: ** **Bandpass filter**

Design a bandpass filter with the following properties:

1. The pass-through wave number should be $\tilde{k} = 0.5$.
2. The bandwidth of the pass-through range should be adjustable.

The filter should be implemented both as a recursive and a non-recursive filter. (Hint: Take the filter [-1 0 2 0 1]/4 as a starting point for the non-recursive implementation. How can you use this filter to obtain a smaller bandwidth?)

4.7 Further Readings

The classical concepts of filtering of discrete time series, especially recursive filters and the z transform are discussed in Oppenheim and Schafer [148] and Proakis and Manolakis [159], 2-D filtering in Lim [124]. A detailed account of nonlinear filters, especially median filters, is given by Huang [83] and Pitas and Venetsanopoulos [155].