Technische Universität Hamburg
Mathematical Image Processing (Winter Term 2022/2023)
Marko Lindner, Kristof Albrecht

Exercise Sheet 2

Deadline: 31.10.22, 12:00pm

Exercise 1 (Interpolation)**.**
In this exercise, we discuss the interpolation of one-dimensional and two-dimensional signals with the help of *interpolation functions*. We call $\phi : \mathbb{R} \to \mathbb{R}$ an **interpolation function**, if it satisfies the conditions

$$\phi(0) = 1 \qquad \text{and} \qquad \phi(i) = 0 \quad \text{for all } i \in \mathbb{Z} \setminus \{0\}.$$

Furthermore, we set

$$\phi^{(i)}(x) := \phi(x - i) \quad \text{for all } x \in \mathbb{R}$$

and for all $i \in \mathbb{Z}$.

a) Let ϕ be an interpolation function and $f = (f_i)_{1 \leq i \leq N} \subset \mathbb{R}^N$ be a one-dimensional signal. Show that the linear combination

$$s_f = \sum_{i=1}^{N} f_i \cdot \phi^{(i)}$$

satisfies the interpolation condition

$$s_f(k) = f_k \qquad \text{for all } k \in \{1, ..., N\}. \tag{1}$$

b) In the setting of part a), consider the space

$$V^{(N)} = \operatorname{span}\{\phi^{(i)} \mid i \in \{1, ..., N\}\}.$$

Prove the following implication: If $s \in V^{(N)}$ satisfies the interpolation condition (1), then we have $s = s_f$. Hence, the interpolation problem is uniquely solveable in $V^{(N)}$.

c) Verify that

$$\phi_0(x) := \chi_{[-0.5,0.5)}(x) = \begin{cases} 1, & \text{if } -0.5 \leq x < 0.5 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\phi_1(x) := \max(1 - |x|, 0)$$

are interpolation functions. Give an example of another interpolation funtion.

d) We can also use interpolation functions in higher dimensions. Given a two-dimensional signal

$$u = (u_{i,j})_{\substack{1 \leq i \leq M \\ 1 \leq j \leq N}} \subset \mathbb{R}^{M \times N}$$

and an interpolation function ϕ, we can define the function

$$s_u(x, y) := \sum_{i=1}^{M} \sum_{j=1}^{N} u_{i,j} \cdot \phi^{(i)}(x) \cdot \phi^{(j)}(y) \qquad \text{for } (x, y) \in \mathbb{R}^2.$$

Show that s_u satisfies the interpolation condition

$$s_u(k, l) = u_{k,l} \qquad \text{for all } (k, l) \in \{1, ..., M\} \times \{1, ..., N\}.$$

Remark: Interpolation with $\phi = \phi_0$ is called *nearest neighbor interpolation*, whereas interpolation with $\phi = \phi_1$ is called *bilinear interpolation*.

Exercise 2 (Image upsampling)**.**
Interpolation can be used to upsample images. In our case, we want to convert an 8-bit image

$$u : \{1, ..., M\} \times \{1, ..., N\} \to \{0, ..., 255\}$$

to an image

$$\tilde{u} : \{1, ..., 2M - 1\} \times \{1, ..., 2N - 1\} \to \{0, ..., 255\}$$

with higher resolution. We proceed as follows:

- Refine the domain of $\Omega = \{1, ..., M\} \times \{1, ..., N\}$ of u to
$$\begin{aligned}\tilde{\Omega} &= \{1, 1.5, ..., M - 0.5, M\} \times \{1, 1.5, ..., N - 0.5, N\} \\ &= \{(1 + i) \cdot 0.5 \mid i \in \{1, ..., 2M - 1\}\} \times \{(1 + j) \cdot 0.5 \mid j \in \{1, ..., 2N - 1\}\}.\end{aligned}$$
- Calculate the missing values of u on $\tilde{\Omega}\backslash\Omega$ via nearest neighbor interpolation or bilinear interpolation (see Exercise 1).

  Attention: Make sure that the result is again in the 8-bit color range.
- The upsampled image is given via
$$\tilde{u}(i, j) := u((1 + i) \cdot 0.5, (1 + j) \cdot 0.5)$$
  for all $(i, j) \in \{1, ..., 2M - 1\} \times \{1, ..., 2N - 1\}$.

Implement this upsampling procedure as a function

```
function [B] = upsampleImg(A, method)
```

where A is the given image and *method* is a string ('nearestneighbor' or 'bilinear') that determines the interpolation method.

Exercise 3 (Image rotation)**.**
Let $u : \{1, ..., M\} \times \{1, ..., N\} \to \{0, ..., 255\}$ be a discrete 8-bit image. If we want to rotate the image by a certain angle $\varphi \in [0, 2\pi)$, we need to solve two subproblems.

a) **Zero padding:** Let D_φ denote the map that rotates the domain $\Omega = \{1, ..., M\} \times \{1, ..., N\}$ around its center by φ. For most angles, we have $D_\varphi(\Omega) \not\subset \Omega$, which would mean that we lose information from our image while simply rotating. Instead, we have to transform the domain Ω via

$$\Omega \longrightarrow \Omega_p = \{1, ..., M + 2p\} \times \{1, ..., N + 2p\}$$

for a sufficiently high padding number $p \in \mathbb{N}$ and set the missing values of Ω_p to 0. Of course, the new image $\tilde{u} : \Omega_p \to \{0, ..., 255\}$ should contain the original image in the center of the new domain Ω_p and the added zeros on the outside.

Write a function

```
function [Ap, p] = padImg(A)
```

that calculates a suitable padding number $p \in \mathbb{N}$ and returns the respective padded version Ap of the initial image A.

b) **Rotation:** Assuming that our image has already been padded with zeros, we can perform the rotation of the image. In order to determine the value

$$\tilde{u}_r(i, j) \qquad \text{for } (i, j) \in \Omega_p$$

of the rotated image $\tilde{u}_r$, we need to reverse the rotation around the center of the image:

$$\tilde{u}_r(i, j) = \tilde{u}(D_{-\varphi}(i, j))$$

In general, we cannot guarantee that $D_{-\varphi}(i, j) \in \mathbb{Z}^2$, which means that we need to apply interpolation methods to determine these values.

Write a function

```
function [B] = rotateImg(A, phi, method)
```

which rotates an already zero-padded image A around its center by an angle *phi*. The string *method* ('nearestneighbor' or 'bilinear') should determine which interpolation method is used. The returned image B should be a 8-bit grayscale image with the same size as A.