Report of Image Restoration Session - Part 1

Seminar Course: Computational Methods for Image Reconstruction

Berfin Kavsut

This report includes the session content with additional explanations and minor corrections compared to the session slides, a retrospective of my session with personal comments, and the code part at the end.

1 Session Content

This session is focused on the digital signal processing techniques used in the image restoration problem. In the first part of the image restoration session that I presented, the blurring effect on images, how to model the blurring effect by using system models, and deblurring approaches were explained. In the second part of the image restoration session, Tom Kratzenberg continued with a focus on the noise effect.

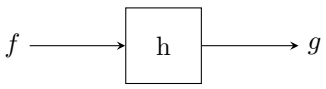
1.1 Image Restoration vs. Image Reconstruction

Jeffrey A. Fessler, who is the author of the book "Image Reconstruction: Algorithms and Analysis", starts his book with the topic of image restoration even though the book is mainly on image reconstruction. He intends to introduce the fundamental mathematical tools for image restoration, which are also common for image reconstruction problems.

The aim of **image reconstruction** is to form an image from measured data that is not interpretable directly as an image, such as a sinogram in tomography. The aim of **image restoration** is to recover the underlying **true object** from the measured image, which is blurry and noisy, in 2D image restoration problems.

1.2 Discrete Measurement Model

Input-output relationships of systems are characterized by **system models**. They relate the unknown quantities, i.e. input, to the observed measurements, i.e. output. This is called **forward model** in the inverse problems field. Here is the diagram to represent the input-output relationship of systems:



Latent image: $f[m,n] \in R : m = 0, \ldots, M-1,\ n = 0, \ldots, N-1$

Measurement: $g[m,n] \in R : m = 0, \ldots, M-1,\ n = 0, \ldots, N-1$

Systems are described by how they respond to impulse functions, which is defined as **impulse response**. In the imaging field, the impulse response is called **point spread function (PSF)**.

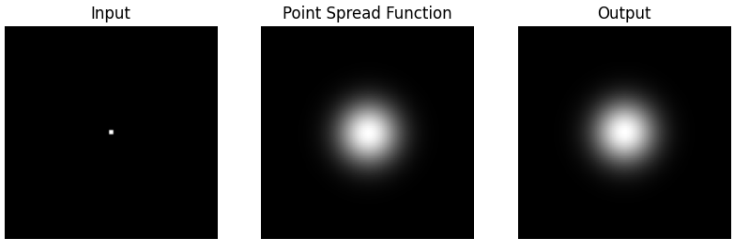


Figure 1: LSI systems

Kronecker impulse function: $\delta_{ij} = \begin{cases} 1 & \text{if } i = 0 \text{ and } j = 0 \\ 0 & \text{otherwise} \end{cases}$

The signals can be written as a linear combination of **impulse functions**. If an input signal is decomposed to its impulse functions, then the output can be described as a linear combination of its impulse responses as well. In a **non-shift invariant system**, the response may vary with the position of the impulse.

Linear combination of shifted impulses:

$$f[m,n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k,l]\delta[m-k, n-l] \tag{1}$$

The equation (1) represents a signal $f[m,n]$ as a linear combination of shifted impulses. In this equation:

- $f[m,n]$ is the signal at the position (m,n).
- $f[k,l]$ represents the amplitude of the signal at the position (k,l).
- $\delta[m-k, n-l]$ is the impulse function shifted to position (k,l).

Linear combination of impulse responses:

$$g[m,n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k,l]b[m,n;k,l] \tag{2}$$

The equation (2) describes the linear combination of shifted impulse responses. In this equation:

- $g[m,n]$ is the output signal.

- $f[k,l]$ represents the input signal at the position (k,l).
- $b[m,n;k,l]$ is the impulse response function at the position (k,l).

1.2.1 Linear-Shift Invariant Systems

Linearity and shift invariance properties make the input-output relationship of systems easier to describe mathematically. Instead of having different impulse responses $b[m,n;k,l]$ for each position at (k,l), the impulse response reduces to $b[m,n]$ independent of the current position of the impulse function. By having only one impulse response, it becomes possible to describe the output of the system for every possible input.

Linearity:
Linearity is defined by two properties: additivity and homogeneity/scaling.

1. **Additivity:** Linear combination of inputs yields linear combination of individual outputs.
   If $f_1[m,n] \xrightarrow{H} g_1[m,n]$ and $f_2[m,n] \xrightarrow{H} g_2[m,n]$, then $f_1[m,n]+f_2[m,n] \xrightarrow{H} g_1[m,n] + g_2m,[n]$.

2. **Homogeneity** or **Scaling:** Scaled input yields scaled output.
   $\alpha f[m,n] \xrightarrow{H} \alpha g[m,n]$

Shift-Invariance: Shifted input yields the shifted version of the output.

$f[m-m_0,n-n_0] \xrightarrow{H} g[m-m_0,n-n_0]$

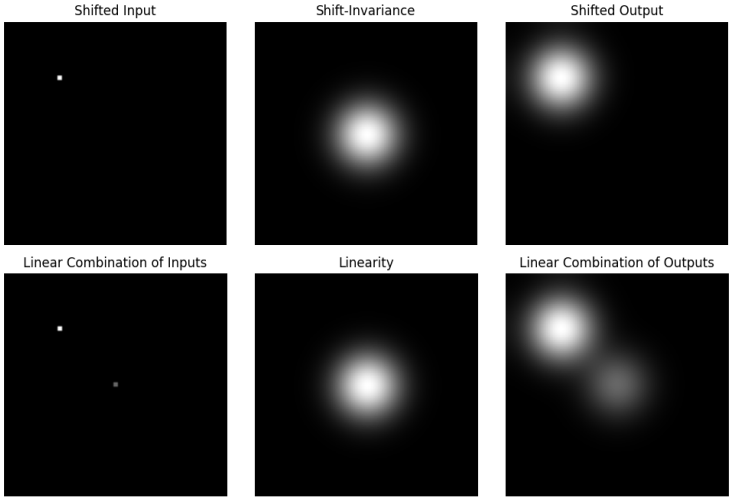


Figure 2: LSI systems

1.2.2 Convolution

When we define the input-output relationship of finite-length discrete signals, there are two common approaches: **zero padding** and **periodization**. The convolution operation is defined as linear convolution with zero padding, and the convolution operation is defined as circular convolution with periodization.

Linear Convolution

When the system model is shift-invariant, where the response to the input at a specific location is determined by the filter's spatial relationship across all positions, then the equation (2) becomes a convolution operation as follows:

$$g[m,n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f[k,l]b[m-k,n-l] = b[m,n] * *f[m,n] \tag{3}$$

Circular Convolution

The linear convolution operation is simplified for periodic signals. The infinite sum at the equation (3) becomes a finite sum. Linear convolution becomes N**-point circular convolution** for N-periodic $f[n]$ input, i.e. $f[n] = f[n \mod N]$,

$$g[n] = \tilde{b}[n]f[n] \sum_{k=0}^{N-1} \tilde{b}[(n-k) \mod N] \cdot f[k], \quad n = 0, \ldots, N-1 \tag{4}$$

where periodic superposition of $b[n]$ is defined by

$$\tilde{b}[n] = \sum_{l} b[n - lN] \tag{5}$$

1.3 Image Restoration Problem: Blur & Noise

We can describe the image restoration problem as the result of a convolution operation with the system's PSF to model the blurring effect caused by the system's impulse response and sum up with the **additive noise** $\epsilon[m,n]$ to reflect the presence of random noise in the image.

$$g[m,n] = b[m,n] * *f[m,n] + \epsilon[m,n] \tag{6}$$

In image restoration problems, the aim is to recover the latent image back with **image denoising** and **image deblurring** methods.

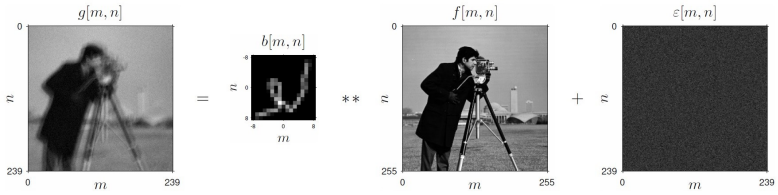


Figure 3: Image restoration

1.4 Continuous-Discrete Model

In most image restoration problems, the unknown **"true object"** is a **continuous-space function**, $f(x, y) : x, y \in R$, so using a discrete-space object and PSF is a simplification.

More realistic continuous-discrete model for image restoration problems are as follows,

$$g[m, n] = \bar{g}[m, n] + \varepsilon[m, n], \quad \bar{g}[m, n] = \iint b(m, n; x, y) f(x, y)\, dx\, dy, \qquad (7)$$

where $b(m, n; x, y)$ denotes the contribution that an impulse object located at (x, y) would make to the expected value of $g[m, n]$. $b(m, n; x, y)$ is the **system model** in this definition.

1.5 Ill-posed problems

Hadamard's Well-Posedness Definition
A mathematical problem is said to be **well-posed** if it satisfies the following three conditions:

1. **Existence:** There exists a solution to the problem.
2. **Uniqueness:** The solution is unique.
3. **Stability:** The solution depends continuously on the data or the input. A small change in the input should result in a small change in the solution.

Non-uniqueness is the main challenge for imaging problems. To make the problem well-posed, we need to impose prior knowledge such as minimum norm estimate.

1.6 Matrix-Vector Representations

Firstly, the Fourier transforms in the discrete space are presented. Later, we will continue with the system matrices, which are used to express convolution with matrix-vector representations.

Fourier transforms, in general, can be understood as projection operations onto a space whose basis functions are complex exponentials. Complex exponentials represent the frequency components of the analyzed signals. Through the projection operation, which is defined by the inner product in the summation, the weight coefficients for each complex exponential can be determined. As a result, the signal can be decomposed into its frequency components and then synthesized back to the original signal.

Discrete Time Fourier Transform (DTFT)
Here is the mathematical tool used to analyze the frequency content of a discrete signal defined in the time domain (1D).

$$b[n] \overset{DTFT}{\longleftrightarrow} B(\Omega) = \sum_{n=-\infty}^{\infty} b[n]e^{-i\Omega n} \tag{8}$$

Convolution Theorem of DTFT
Convolution in the time domain (1D) is related to *multiplication in the frequency domain.*

$$g[n] = b[n] * f[n] \overset{DTFT}{\longleftrightarrow} \bar{G}(\Omega) = B(\Omega) \cdot F(\Omega) \tag{9}$$

Discrete Space Fourier Transform (DSFT)
Here is the mathematical tool used to analyze the frequency content of a discrete signal defined in a spatial domain (2D).

$$B(\Omega_1, \Omega_2) = \sum_{m,n} b[m,n] \cdot e^{-i(\Omega_1 m + \Omega_2 n)} \tag{10}$$

Convolution Theorem of DSFT
Convolution in the spatial domain (2D) is related to *multiplication in the frequency domain.*

$$\bar{g}[m,n] = b[m,n] * *f[m,n] \overset{DSFT}{\longleftrightarrow} \bar{G}(\Omega_1, \Omega_2) = B(\Omega_1, \Omega_2) \cdot F(\Omega_1, \Omega_2) \tag{11}$$

1.6.1 1D Matrix-Vector Representation

Consider a 1D convolution relationship for an input signal $f[n]$ as follows:

$$g[n] = b[n] * f[n] + \varepsilon[n] = \sum_{k=0}^{N-1} b[n-k] \cdot f[k] + \varepsilon[n], \quad n = 0, \ldots, N-1 \tag{12}$$

We want to represent the preceding "DSP-style" formula in this matrix-vector form,

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\varepsilon} \tag{13}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} g[0] \\ \vdots \\ g[N-1] \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} f[0] \\ \vdots \\ f[N-1] \end{bmatrix}$$

The equation (12) turns into the matrix representation of the convolution operation.

$$\underbrace{\begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ y[N-1] \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} b[0] & 0 & \dots & 0 \\ b[1] & b[0] & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ b[N-1] & b[N-2] & \dots & b[0] \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[M-1] \end{bmatrix}}_{\mathbf{x}} \tag{14}$$

$\mathbf{A}$ is a matrix defining the convolution operation, i.e. ***system matrix***, $\mathbf{x}$ is the ***input vector***, and $\boldsymbol{\varepsilon}$ is the ***error*** or ***noise vector***.

It is an engineering decision to choose the end conditions of the system matrix $\mathbf{A}$. We will talk about 3 cases of end conditions: (1) zero end condition, (2) extended end condition, and (3) periodic end condition.

1.6.2 End-Conditions: Zero

Assumption: $f[n]$ is zero for $n < 0$ and $n \geq N$, using the zero end condition or *Dirichlet boundary condition*. This end condition corresponds to the linear convolution where we used zero padding approach to convert the infinite sum to a finite sum.

Matrix-Vector Representation:

$$\mathbf{A} = \begin{bmatrix} b[0] & b[-1] & 0 & \cdots & 0 \\ b[1] & b[0] & b[-1] & \cdots & 0 \\ 0 & b[1] & b[0] & b[-1] & \vdots \\ \vdots & & & \ddots & \\ 0 & \cdots & 0 & b[1] & b[0] \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} f[0] \\ \vdots \\ f[N-1] \end{bmatrix} \tag{15}$$

NxN system matrix $\mathbf{A}$ is **Toeplitz** (constant along the diagonals). The elements a_{ij} of matrix $\mathbf{A}$ are connected to a general impulse response function $b[n]$ through the relation $a_{ij} = b[i-j], \quad i, j = 1, \dots, N$.

In shift-invariant problems, the matrix-vector representation is often used for mathematical *analysis* convenience rather than direct *implementation*. In shift-variant problems, storing A as a matrix is frequently appropriate. In such cases, a **sparse matrix** representation is often preferred for efficient implementation.

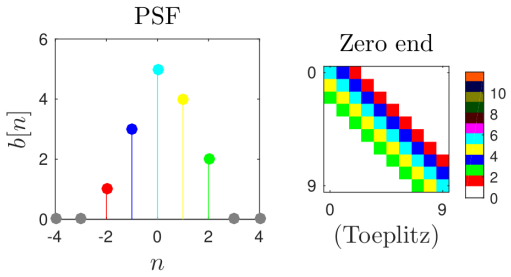


Figure 4: System matrix with zero end condition (1D)

1.6.3 End conditions: Extended

In many situations, e.g., optical imaging, the measurements are influenced by a larger scene than the field of view of the aperture due to the spreading caused by the imaging system PSF.

Extended end conditions are often **more realistic** for restoration problems and should be used when feasible. However, often engineers may use **approximations** like "replicated," "mirror," and "periodic" end conditions to save computation. Similar to zero end conditions, again the matrix representation of convolution is more useful for *analysis* than for *computation.*

Matrix-Vector Representation: A is a $Nx(N+L-1)$ rectangular matrix where L is the length of the impulse response ($L = 3$ for this particular $b[n]$).

$$\mathbf{A} = \begin{bmatrix} b[1] & b[0] & b[-1] & 0 & \cdots & 0 \\ 0 & b[1] & b[0] & b[-1] & \cdots & 0 \\ & & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & b[1] & b[0] & b[-1] \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} f[-1] \\ f[0] \\ \vdots \\ f[N-1] \\ f[N] \end{bmatrix} \tag{16}$$

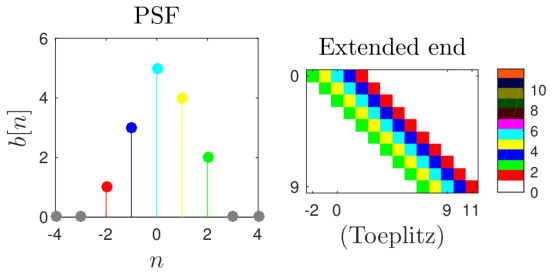


Figure 5: System matrix with extended end condition (1D)

1.6.4 End conditions: Periodic

Assumption: $f[n]$ is N-periodic, i.e. $f[n] = f[n \mod N]$. This end condition corresponds to the circular convolution where we used the periodization approach to convert the infinite sum to a finite sum.

Firstly, we need to periodize the blur kernel by **periodic superposition:**

$$\tilde{b}[n] = \sum_{l} b[n - lN] \tag{17}$$

Matrix-Vector Representation: NxN system matrix A is **periodic/circulant**.

$$\mathbf{A} = \begin{bmatrix} \tilde{b}[0] & \tilde{b}[N-1] & \tilde{b}[N-2] & \cdots & \tilde{b}[2] & \tilde{b}[1] \\ \tilde{b}[1] & \tilde{b}[0] & \tilde{b}[N-1] & \tilde{b}[N-2] & \cdots & \tilde{b}[2] \\ & & & \ddots & & \\ \tilde{b}[N-1] & \tilde{b}[N-2] & \tilde{b}[N-3] & \cdots & \tilde{b}[1] & \tilde{b}[0] \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} f[0] \\ \vdots \\ f[N-1] \end{bmatrix}$$

where $a_{ij} = \tilde{b}[(i-j) \mod N], \quad i, j = 1, \ldots, N.$

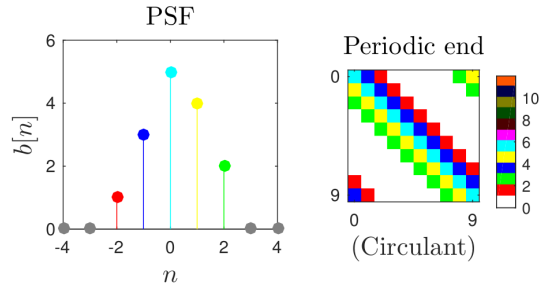


Figure 6: System matrix with periodic end condition (1D)

1.6.5 2D Matrix-Vector Representation

One can use the matrix-vector representation for 2D systems just as used for 1D systems. The necessary steps are to vectorize the image to have a vector representation for images by the operation vec : $R^{M\times N} \to R^{MN}$ such that $\mathbf{y} = \text{vec}(\mathbf{g})$, and then define the system matrix accordingly. For vectorization, **lexicographic ordering** is utilized:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{MN} \end{bmatrix}, \quad y_i = g[m(i), n(i)], \quad i = 1, \ldots, MN \tag{18}$$

Vector index i maps to pixel coordinates $[m(i), n(i)]$ as $m(i) = (i-1) \mod M$, $\quad n(i) = \left\lfloor \frac{i-1}{M} \right\rfloor$. It is easier to track the indices by looking at Figure 7.

$y_1 = g[0,0]$	$y_2 = g[1,0]$	$\ldots$	$y_M = g[M-1,0]$
$y_{M+1} = g[0,1]$	$y_{M+2} = g[1,1]$	$\ldots$	$y_{2M} = g[M-1,1]$
		$\vdots$	
$y_{M(N-1)+1} = g[0,N-1]$	$y_{M(N-1)+2} = g[1,N-1]$	$\ldots$	$y_{MN} = g[M-1,N-1]$

$\to m$
$\downarrow$
n

Figure 7: Vectorization

1.6.6 Representing the Matrix A as Block Matrix

By vectorization, the system matrix $\mathbf{A}$ has a block matrix form,

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{0,0} & \cdots & \mathbf{A}_{0,N-1} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{N-1,0} & \cdots & \mathbf{A}_{N-1,N-1} \end{bmatrix} \tag{19}$$

where the $M \times M$ submatrix $\mathbf{A}_{nl}$ describes how the l-th row of the input image contributes to the n-th row of the output image and has elements $[\mathbf{A}_{nl}]_{mk} = b[m, n; k, l]$.

1.7 2D End conditions: Zero

For 2D linear shift-invariant systems, $\mathbf{A}$ has the block form where the submatrices have elements $[\mathbf{A}_{nl}]_{mk} = b[m-k, n-l]$.

Because of the $(m-k)$ dependence, each of the blocks is **Toeplitz** in this shift invariant case, so $\mathbf{A}$ is said to have **Toeplitz blocks**. Furthermore, because of the $(n-l)$ dependence, all of the blocks along each "diagonal" in the block form are the same, so $\mathbf{A}$ is said to be **block Toeplitz**. Combined, we say any such $\mathbf{A}$ is **block Toeplitz with Toeplitz blocks (BTTB).**

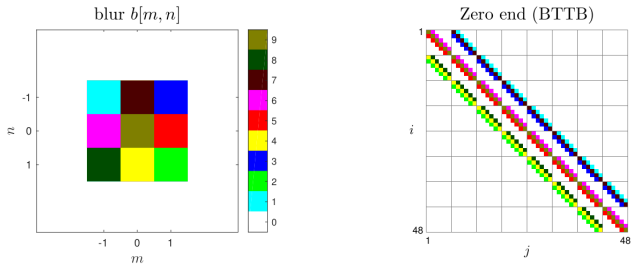


Figure 8: System matrix with zero end condition (2D)

1.7.1 2D End conditions: Extended

Similar to the zero end condition, the system matrix of the extended end condition also has a BTTB form. Toeplitz blocks are larger to capture all effects coming from the additional columns to zero end conditions. Block Toeplitz is again larger to capture all blur effects coming from the additional rows to zero end condition.

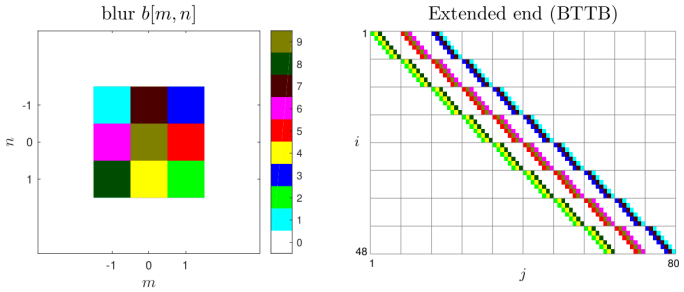


Figure 9: System matrix with extended end condition (2D)

1.7.2 2D End conditions: Periodic

The system matrix for periodic end condition has a form of **Block Circulant with Circulant Blocks (BCCB)**. Each block is circulant due to the periodization in the columns. The block is circulant due to the periodization in the rows.

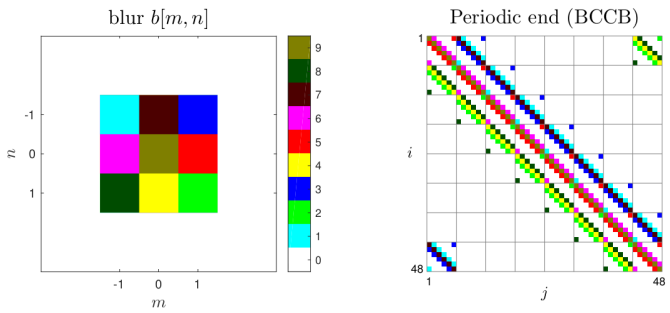


Figure 10: System matrix with periodic end condition (2D)

1.8 Circulant Analysis of Shift-Invariant Blur

Analysis using **circulant matrices** helps us relate matrix algebra solutions to signal processing principles. We reduce computation by replacing large matrix operations with simpler **fast Fourier transform (FFT)** calculations.

The link between **circulant matrices** and **circular convolution** is the convolution property of the **discrete Fourier transform (DFT)**.

1.8.1 DFT, Circular Convolution, & Periodic Signals

The N-point DFT of $b[n]$ is defined as

$$b[n] \quad \overset{DFT}{\longleftrightarrow} \quad B_k = \sum_{n=0}^{N-1} b[n] e^{-i\frac{2\pi}{N}kn}, \quad n,k = 0,\ldots,N-1 \tag{20}$$

Here, B_k is an N-length sequence. Similar to the matrix-vector representation of convolution, the summation operation of the DFT can be represented as matrix-vector multiplication by $\vec{\mathbf{B}} = \mathbf{Qb}$ where $\mathbf{Q}$ is the $N \times N$ **DFT matrix** .

$$Q = \begin{bmatrix} \omega^{0\cdot 0} & \omega^{0\cdot 1} & \omega^{0\cdot 2} & \cdots & \omega^{0\cdot(N-1)} \\ \omega^{1\cdot 0} & \omega^{1\cdot 1} & \omega^{1\cdot 2} & \cdots & \omega^{1\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{(N-1)\cdot 0} & \omega^{(N-1)\cdot 1} & \omega^{(N-1)\cdot 2} & \cdots & \omega^{(N-1)\cdot(N-1)} \end{bmatrix}, \quad \omega = e^{-2\pi i/N} \tag{21}$$

Convolution Theorem of DFT:

$$\bar{g}[n] = b[n]_N f[n] \quad \overset{DFT}{\longleftrightarrow} \quad \bar{G}_k = B_k F_k, \quad n,k = 0,\ldots,N-1 \tag{22}$$

1.8.2 DFT vs. DTFT

DTFT is the Fourier transform of *non-periodic* and *discrete* signals and the Fourier transform yields *periodic* (due to discreteness) and *continuous* (due to

non-periodicity) transformation results. DTFT is the Fourier transform of *periodic* and *discrete* signals and the Fourier transform yields periodic (due to discreteness) and discrete (due to periodicity) transformation results. The connection between DFT and DTFT is that DFT can be viewed as a sampled version of DTFT:

$$b[n] \overset{DFT}{\longleftrightarrow} B_k = \sum_{n=0}^{N-1} b[n] e^{-i\frac{2\pi}{N}kn} = B(\Omega)\Big|_{\Omega=\frac{2\pi}{N}k}, \quad k = 0, \ldots, N-1 \tag{23}$$

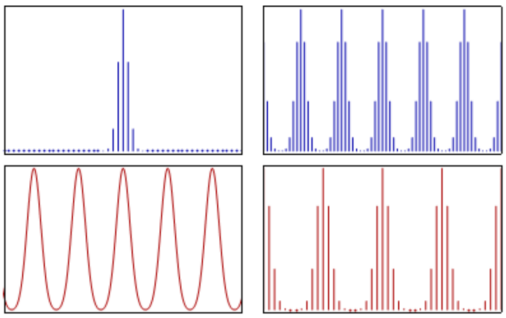


Figure 11: **Top:** Non-periodic and periodic signals, **Bottom:** Fourier transforms

1.8.3 Circulant Analysis

By the convolution theorem as described in the equation (22), one can write the input-output relationship for the system in the frequency domain as follows:

$$\underbrace{\begin{bmatrix} \bar{G}_0 \\ \vdots \\ \bar{G}_{N-1} \end{bmatrix}}_{\text{N x 1}} = \underbrace{\begin{bmatrix} B_0 & \cdots & 0 \\ & \ddots & \\ 0 & & B_{N-1} \end{bmatrix}}_{\text{N x N}} \underbrace{\begin{bmatrix} F_0 \\ \vdots \\ F_{N-1} \end{bmatrix}}_{\text{N x 1}}, \text{i.e.} \quad \vec{\mathbf{G}} = \mathbf{\Gamma}\vec{\mathbf{F}} \tag{24}$$

Here, $\vec{\mathbf{G}} = \mathbf{Q}\mathbf{y}$ and $\vec{\mathbf{F}} = \mathbf{Q}\mathbf{x}$, then we can state that $\vec{\mathbf{y}} = \mathbf{Q^{-1}\Gamma Q x}$. The inverse of DFT matrix $\mathbf{Q^{-1}} = \frac{1}{N}\mathbf{Q'}$, where $\mathbf{Q'}$ is the Hermitian transpose, i.e. conjugate transpose, of $\mathbf{Q}$.

Any circulant matrix, e.g. the circulant system matrix for periodic end conditions, has the following matrix decomposition:

$$\mathbf{A} = \mathbf{Q^{-1}\Gamma Q} = \frac{1}{N}\mathbf{Q'\Gamma Q} \tag{25}$$

Consider a circulant matrix $\mathbf{A}$ with eigenvalues B_k. The eigenvector decomposition can be expressed as:

$$\mathbf{A} = \mathbf{Q^{-1}\Gamma Q} \tag{26}$$

where Q is the DFT matrix and $\mathbf{\Gamma}$ is a diagonal matrix with the eigenvalues B_k. When $\mathbf{A}$ is a circularly shift-invariant filter, frequency response is embedded in the diagonal elements of $\mathbf{\Gamma}$.

1.9 Simple Image Restoration Problems

1.9.1 The Deconvolution Solution

The **convolution property** of the Fourier transform, where $G(\Omega_1, \Omega_2) = B(\Omega_1, \Omega_2)F(\Omega_1, \Omega_2)$, suggests the following **inverse-filter** solution:

$$\hat{F}(\Omega_1, \Omega_2) = \begin{cases} \frac{G(\Omega_1,\Omega_2)}{B(\Omega_1,\Omega_2)}, & \text{if } B(\Omega_1, \Omega_2) \neq 0 \\ 0, & \text{if } B(\Omega_1, \Omega_2) = 0 \end{cases} \tag{27}$$

Equivalently, in the spatial domain: $\hat{f}[m, n] = b_{\text{inv}}[m, n] ** g[m, n]$ where $b_{\text{inv}}[m, n]$ is the inverse Fourier transform of $\frac{1}{B(\Omega_1,\Omega_2)}$.

1.9.2 Matrix Inverse Solution

Algebraic reconstruction techniques are based on linear algebra concepts. From the matrix-vector representation of system functions:

$$\hat{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{y} \tag{28}$$

Circulant Matrix Case:

For initial understanding, consider the case where $\mathbf{A}$ is **circulant**. Then, $\mathbf{A}^{-1} = \mathbf{Q}^{-1}\mathbf{\Gamma}^{-1}\mathbf{Q}$, and the solution becomes:

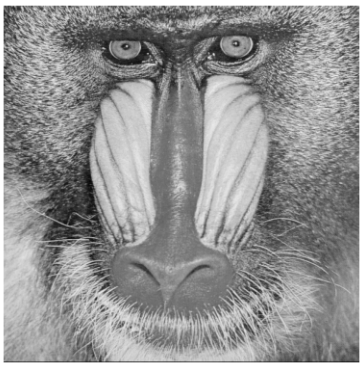
$$\hat{\mathbf{x}} = \mathbf{Q}^{-1}\mathbf{\Gamma}^{-1}\mathbf{Q}\mathbf{y} \tag{29}$$

$\mathbf{Q}$ corresponds to the DFT matrix, and $\mathbf{\Gamma}^{-1}$ has reciprocals of samples of the system frequency response $B(\Omega)$ along its diagonal.

2 Code Exercise

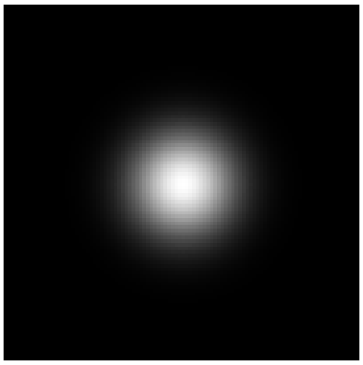
2.1 Read the Image

```
# import python libraries
from skimage import color

import matplotlib.pyplot as plt
import numpy  as np
import scipy.ndimage as nd

# display the input image
# read the image
img_path = "images/baboon.png"
img = plt.imread(img_path)

# apply min-max normalization
img = color.rgb2gray(img)
img = (img - img.min()) / (img.max() - img.min())
```



2.2 Create Gaussian Kernel

```
# Gaussian filter in spatial domain
def gaussian_kernel(N, sigma):
    gaussian = lambda x, y: (1/(2*np.pi*sigma**2)) * np.exp(-(x**2 + y**2)/(2*sig

    x = (1/N) * np.linspace(-N//2, N//2, N, endpoint=False)
    y = (1/N) * np.linspace(-N//2, N//2, N, endpoint=False)
    xx, yy = np.meshgrid(x, y, indexing='ij')

    gaussian_kernel = (1/N**2) * gaussian(xx, yy)
    return gaussian_kernel

kernel = gaussian_kernel(N=100, sigma=0.1)
```

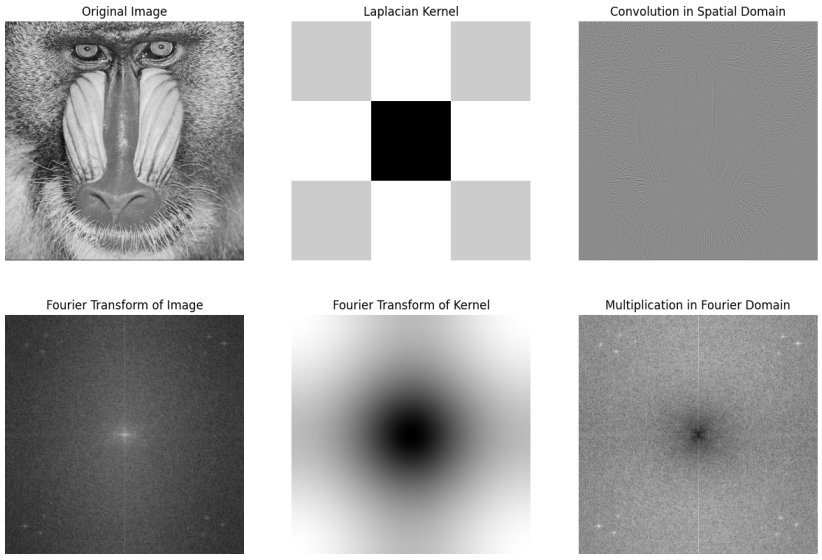


2.3 LSI Systems and Convolution

```
# Impulse function
imgs = np.zeros(shape=(3,size,size))
imgs[0][49:51, 49:51] = 1

# Shift-invariance
imgs[1][24:26, 24:26] = 1

# Linearity
coeff = [2, 5]
imgs[2] = coeff[0] * imgs[0] + coeff[1] * imgs[1]

# Convolution with Gaussian kernel
imgs_conv = [nd.convolve(img, kernel) for img in imgs]
```

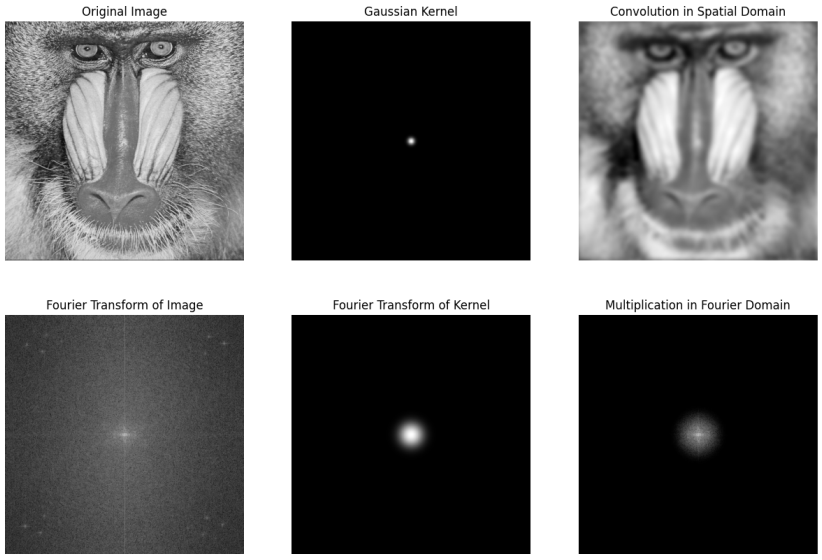
2.4 Convolution Theorem: Example with High Pass Filter

```
# Laplacian kernel
kernel = np.array([[0,  1, 0],
                   [1, -4, 1],
                   [0,  1, 0]])

# Fourier transform of the image
img_ft = np.fft.fft2(img)
img_ft = np.fft.fftshift(img_ft)

# Zero-padded kernel
# Fourier transform of the kernel has the same size as the input image
h, w = img.shape
kh, kw = kernel.shape
kernel_pad = np.zeros_like(img, dtype="float64")
kernel_pad[h//2-kh//2:h//2+kh//2+1, w//2-kw//2:w//2+kw//2+1] = kernel

# Fourier transform of the kernel
kernel_ft = np.fft.fft2(kernel_pad)
kernel_ft = np.fft.fftshift(kernel_ft)

# Filter the image in Fourier domain
img_filtered_ft = kernel_ft * img_ft
img_filtered = np.fft.ifft2(np.fft.fftshift(img_filtered_ft))
img_filtered = np.fft.fftshift(img_filtered).real
```



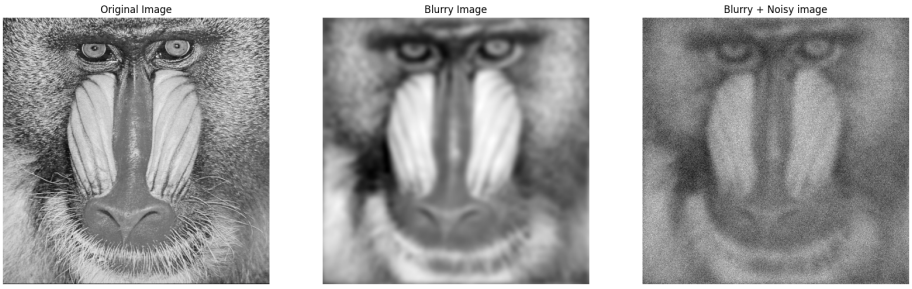
2.5 Convolution Theorem: Example with Low Pass Filter

```
# Gaussian filter in the spatial domain
kernel = gaussian_kernel(N=img.shape[0], sigma=0.01)

# Gaussian filter in the frequency domain
kernel_ft = np.fft.fft2(kernel)
kernel_ft = np.fft.fftshift(kernel_ft)

# Image in the frequency domain
img_ft = np.fft.fft2(img)
img_ft = np.fft.fftshift(img_ft)

# Filter the image in frequency domain
img_filtered_ft = kernel_ft * img_ft
img_filtered = np.fft.ifft2(np.fft.fftshift(img_filtered_ft))
img_filtered = np.fft.fftshift(img_filtered).real
```



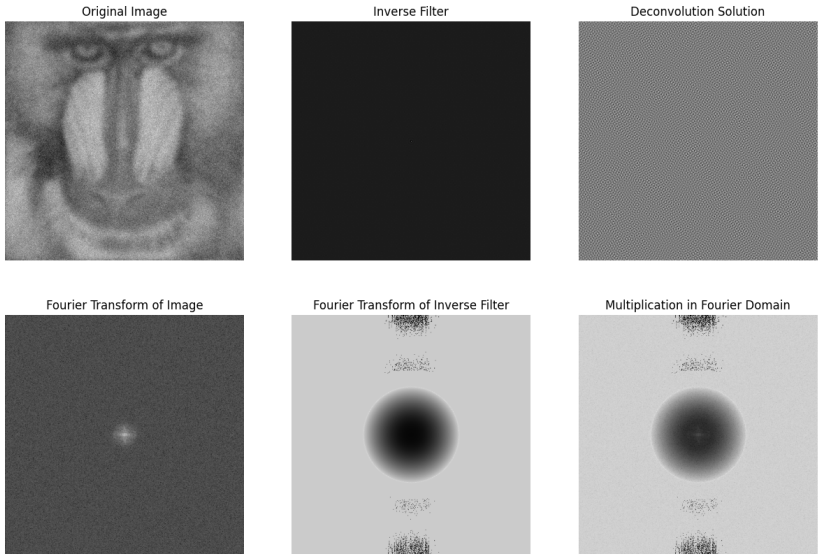
2.6 Image Restoration Problem

```
# Blur the image by applying Gaussian filter to the image
# Take the result from the previous code
img_blurry = img_filtered

# Add Gaussian noise on the blurry image
img_blurry_noisy = img_blurry + np.random.normal(0., 0.1, img.shape)
```



2.7 Deconvolution Solution for Blurry & Noisy

```
# Inverse filter in the frequency domain
h_ft = kernel_ft
h_inv_ft = np.where(h_ft, 1 / (h_ft + 1e-9), 0)

# Inverse filter in spatial domain
h_inv = np.fft.ifft2(np.fft.fftshift(h_inv_ft))
h_inv = np.fft.fftshift(h_inv)

# Fourier transform of the image
img_blurry_noisy_ft = np.fft.fft2(img_blurry_noisy)
img_blurry_noisy_ft = np.fft.fftshift(img_blurry_noisy_ft)

# Deconvolution in frequency domain
img_restored_ft = h_inv_ft * img_blurry_noisy_ft
img_restored = np.fft.ifft2(np.fft.fftshift(img_restored_ft))
img_restored = np.fft.fftshift(img_restored).real
```



2.8 1-D Matrix-Vector Representation

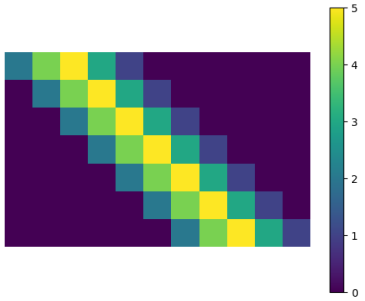
```
import numpy as np

x = np.arange(0, 7); N = len(x)
h = np.array([1, 3, 5, 4, 2]); L = len(h)
indices = np.array([-2, -1, 0, 1, 2])

h_orig = h
h = np.zeros_like(h_orig)
h[indices] = h_orig
```

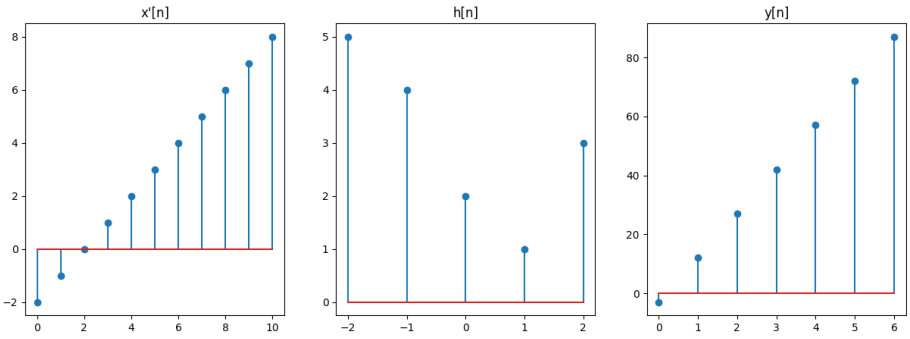
2.9 Define Transformation Matrix from 1D PSF

2.9.1 End Conditions: Extended End Condition

```
# Construct the system matrix with extended end condition
def system_matrix_extended(h, N, L):
    A = np.zeros(shape=(N, N+L-1))
    for i in range(N):
        for k, j in enumerate(indices):
            A[i, i+k] = h[-j]
    return A

A = system_matrix_extended(h=h, N=N, L=L)
```

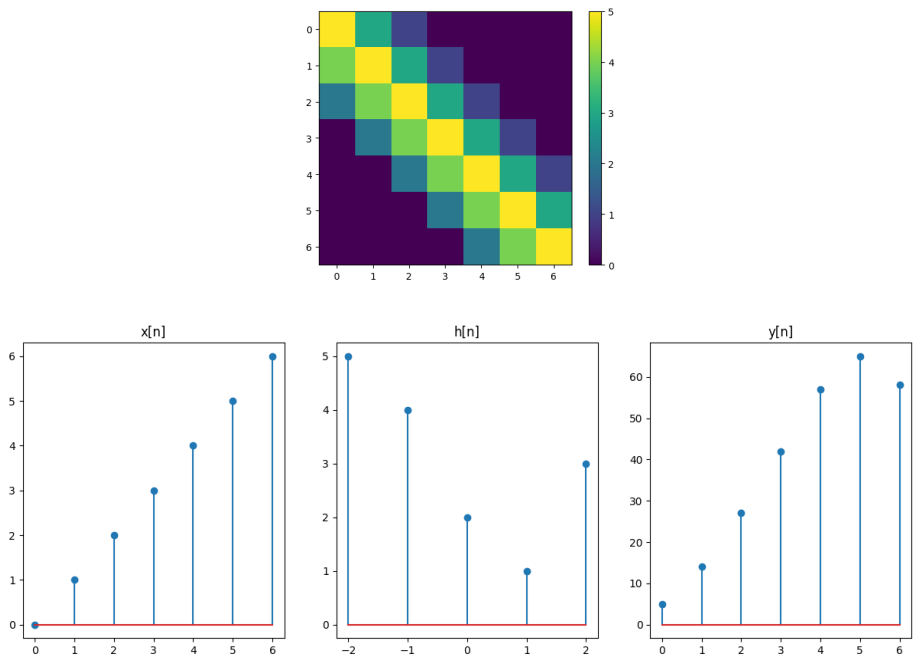
```
# Find the output with matrix-vector multiplication
x_extended = np.array([-2, -1, *x, 7, 8])
y = A @ x_extended
```





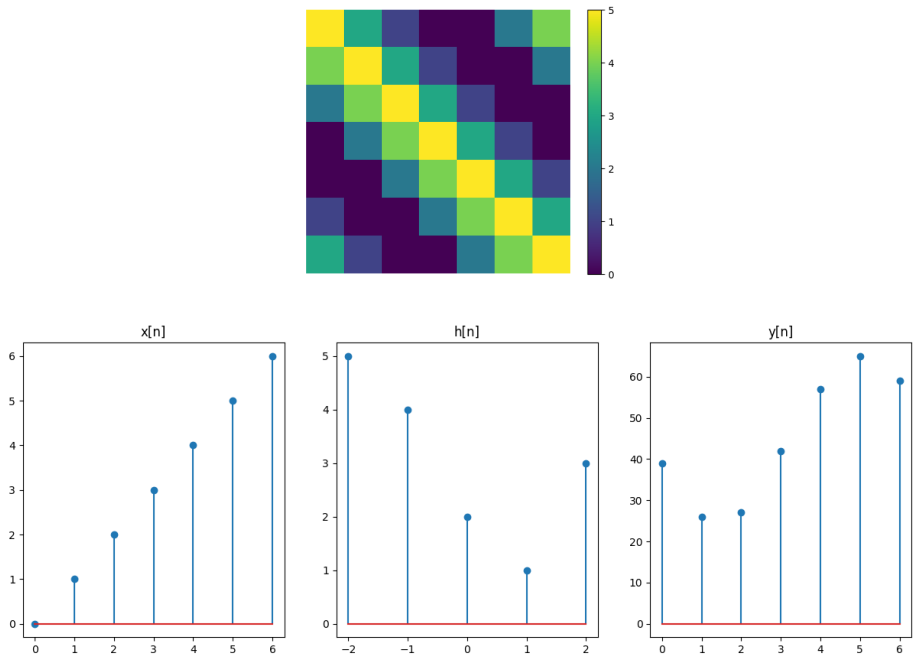
2.9.2 End Conditions: Zero End Condition

```
# Construct the system matrix with zero end condition
def system_matrix_zero(h, N, L):
    A = np.zeros(shape=(N, N+L-1))
    for i in range(N):
        for k, j in enumerate(indices):
            A[i, i+k] = h[-j]

    A = A[:, L//2:-(L//2)]
    return A

A = system_matrix_zero(h=h, N=N, L=L)

# Find the output with matrix-vector multiplication
y = A @ x
```



2.9.3 End Conditions: Periodic End Condition, Example 1

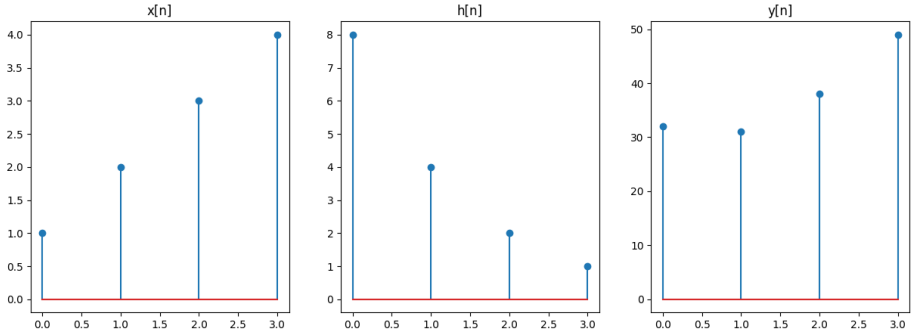
```
# Construct the system matrix with periodic end condition
def system_matrix_periodic(h, N):
    A = np.zeros(shape=(N, N))

    for i in range(N):
        for k in indices:
            A[i, (i + k) % N] = h[-k]
    return A

A = system_matrix_periodic(h=h, N=N)

print(f'System matrix A: N x (N+L-1) = {A.shape[0]} x {A.shape[1]}')

y = A @ x
```



2.9.4 End Conditions: Periodic End Condition, Example 2

```
# Input signal
x = np.array([1, 2, 3, 4])

N = len(x)
M = len(h)

# Periodic superposition
h_circ = np.zeros(N)
for i in range(M):
    h_circ[i % N] += h[i]

# Construct the system matrix with periodic end condition
def system_matrix_periodic(h, N):
    A = np.zeros((N, N))

    # Periodic superposition
    M = len(h)
    h_circ = np.zeros(N)
    for i in range(M):
        h_circ[i % N] += h[i]
```

```
    # System matrix
    for i in range(N):
        for j in range(N):
            A[i, j] = h_circ[(i - j) % N]

    return A

A = system_matrix_periodic(h, N)

y = A @ x
```



2.10 Circulant analysis in 1D

```
# Construct the discrete Fourier transform (DFT) matrix
def dft_matrix(N):
    omega = np.exp(-2j * np.pi / N)
    W = np.array([[omega**(i * j) for j in range(N)] for i in range(N)])

    # inverse of W, which is complex conjugate transpose of W
    W_inv = (1 / N) * np.conj(W.T)
    return W, W_inv
```

```
# Input signal
x = np.array([1, 2, 3, 4])

# Compute DFT of x
W, W_inv = dft_matrix(N=4)
X = W @ x

# Sanity check for the DFT matrix computation
x_ft = np.fft.fft(x)

assert np.allclose(X.real, x_ft.real)
assert np.allclose(X.imag, x_ft.imag)
print("Computation of DFT matrix is correct!")

# Compute circular convolution of x with h by using system matrix
A = system_matrix_periodic(h, N=x.shape[0])
y_conv = A @ x

# Compute circular convolution by using DFT matrix
H = W @ h_circ
Y = H * X
y_dft = W_inv @ Y
y_dft = y_dft.real

# Sanity check for the system matrix with periodic end condition
assert np.allclose(y_dft, y_conv)
print("Circular convolution with periodic system matrix is correct!")

# Eigendecomposition of the system matrix A, which is a circulant matrix with pe
eigenvalues, eigenvectors = np.linalg.eig(A)

np.set_printoptions(precision=2, suppress=True)

print("Eigenvalues of circulant matrix A: ", eigenvalues)
print("Discrete Fourier transform of h: ", H)
```

2.11 Matrix-Inverse Solution using Circulant Analysis in 1D

```
A_ = W_inv @ np.diag(H) @ W

assert np.allclose(A, A_)
print("We could build the inverse of system matrix A by DFT method!")
```

```
A_inv = W_inv @ np.linalg.inv(np.diag(H)) @ W
x_hat = (A_inv @ y).real

assert np.allclose(x, x_hat)
print("We could restore x from the output signal y!")
```

2.12 2-D Matrix Vector Representation

2.12.1 Lexicographic Ordering: Vectorization

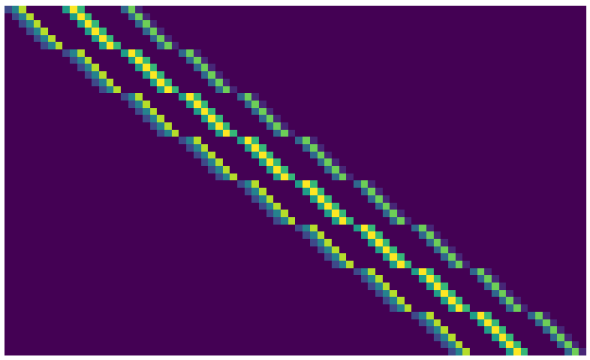
```
# Vectorization for an image
import numpy as np

image = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

def vec(image):
    N, M = image.shape
    y = np.zeros(shape=(M*N,))

    for m in range(M):
        for n in range(N):
            y[m+n*M] = image[n, m]

    return y

# Devectorization of an image vector
def devec(y, M, N):
    I = np.zeros(shape=(N,M))

    for n in range(N):  # iterate over rows
        for m in range(M): # iterate over columns
            I[n,m] = y[m+n*M]
    return I

y = vec(image)
I = devec(y, M=3, N=3)
```

2.13 Define Transformation Matrix from 2D PSF

```
# Define 2D PSF
psf = np.array([[1, 7, 3],
                [6, 9, 5],
                [8, 4, 2]])
```

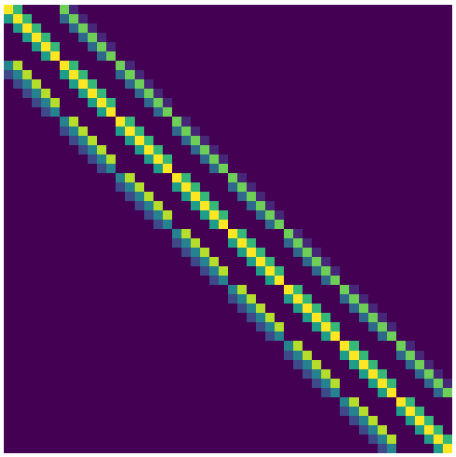
2.13.1 End Conditions: Extended

```
# Construct the system matrix for 2D PSF with extended end conditions
def system_matrix_extended_2d(psf, M, N):
    BM, BN = psf.shape

    A = np.zeros(shape=(M*N, (M+2)*(N+2)))
    psf_flipped = np.flip(np.flip(psf, axis=0), axis=1)

    for n in range(N):
        for l in range(BN):
            psf_row = psf_flipped[l, :].reshape(1, BM)

            block = np.zeros(shape=(M, (M+2)))
            for m in range(M):
                block[m, m:m+BM] = psf_row

            r = n*M
            c = (n+l)*(M+2)

            A[r:(r+M), c:(c+M+2)] = block
    return A

M = 6  # column number i.e. row length
N = 8  # row number i.e. column length

A = system_matrix_extended_2d(psf, M=M, N=N)
```



2.13.2 End Conditions: Zero

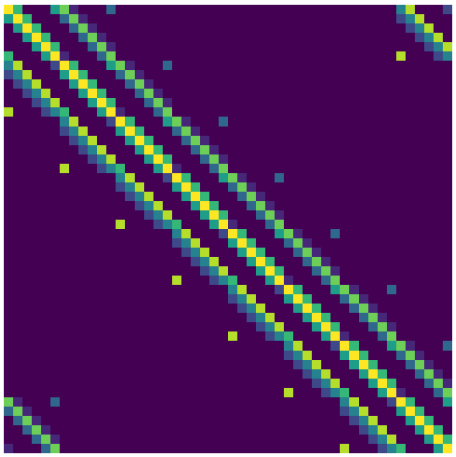
```
# Construct the system matrix for 2D PSF with zero end condition
def system_matrix_zero_2d(psf, M, N):
    BM, BN = psf.shape

    A = np.zeros(shape=(M*N, M*(N+2)))
    psf_flipped = np.flip(np.flip(psf, axis=0), axis=1)

    for n in range(N):
        for l in range(BN):
            psf_row = psf_flipped[l, :].reshape(1, BM)

            block = np.zeros(shape=(M, (M+2)))
            for m in range(M):
                block[m, m:m+BM] = psf_row

            # cut the blocks for zero end condition
            block = block[:, BM//2:-(BM//2)]

            r = n*M
            c = (n+l)*M

            A[r:(r+M), c:(c+M)] = block

    # cut the system matrix for zero end condition
    A = A[:, M:-M]
    return A

A = system_matrix_zero_2d(psf, M=6, N=8)
```



2.13.3 End Conditions: Periodic

```
# Construct the system matrix for 2D PSF with periodic end condition
psf = np.array([[1, 7, 3],
                [6, 9, 5],
                [8, 4, 2]])

def system_matrix_periodic_2d(psf, M, N):

    # rearrange the kernel according to the indices
    h = np.zeros_like(psf)
    for n, i in enumerate([-1, 0, 1]):
        for m, j in enumerate([-1, 0, 1]):
            h[i,j] = psf[n,m]

    BM, BN = h.shape

    # A is a block circulant with circulant blocks
    A = np.zeros(shape=(M*N, M*N))

    for n in range(N):
        for l in [-1, 0, 1]:
            h_row = h[l, :].reshape(BM)

            # create circulant block
            block = np.zeros(shape=(M, M))
            for m in range(M):
                for k in [-1, 0, 1]:
                    # circular convolution by (m + k) % M
                    # flipping by -k
```

```
                block[m, (m + k) % M] = h_row[-k]

            # put the block to the corresponding block location
            r = n*M

            # flipping the order of blocks by -l
            c = ((n-l) % N) * M
            A[r:(r+M), c:(c+M)] = block

    return A

A = system_matrix_periodic_2d(psf, M=6, N=8)
```



2.14 Matrix inverse solution in 2D

```
M, N = (6, 8)

x_img = np.random.normal(size=(N,M))

A = system_matrix_periodic_2d(psf, M=M, N=N)
x = vec(x_img)
y = A @ x

A_inv = np.linalg.inv(A)
x_hat = A_inv @ y
x_img_hat = devec(x_hat, M=M, N=N)

assert np.allclose(x_hat, x)
print("We could restore input image!")
```