COL783 A2

Anuj Naval (2021CS50591) and Harshit Gautam (2021CS50129)

October 2025

1 Question 1

Part (a)

If f is the original image sampled with spacing ΔX and f' is obtained by resampling f at new locations (zoom/shrink) using interpolation, then the Fourier transform of f' is:

$$f'(x', y') = f\Big(\frac{x'}{k}, \frac{y'}{k}\Big) \quad \Longrightarrow \quad F'(u, v) = \frac{1}{k^2} F\Big(\frac{u}{k}, \frac{v}{k}\Big)$$

where $F(u, v)$ is the 2D Fourier transform of f.

- For zooming ($k > 1$), $F'(u, v)$ is a compressed version of $F(u, v)$ in the frequency domain.
- For shrinking ($0 < k < 1$), $F'(u, v)$ is expanded, which may lead to aliasing if high frequencies are not removed.

In our custom code, we perform resampling using nearest-neighbor downsampling and upsampling, which approximates this Fourier-domain relationship.

Part (b)

To avoid aliasing when shrinking ($0 < k < 1$), a low-pass filter must be applied before resampling. The ideal frequency-domain filter is a rectangular (box) filter:

$$H(u, v) = \begin{cases} 1, & |u| \leq \frac{k}{2} f_s,\ |v| \leq \frac{k}{2} f_s \\ 0, & \text{otherwise} \end{cases}$$

where $f_s = 1/\Delta X$ is the sampling frequency.
The corresponding spatial-domain kernel is the 2D sinc function:

$$h(x, y) = k^2 \operatorname{sinc}(kx) \operatorname{sinc}(ky)$$

In our code, the function `ideal_rect_lowpass_fft(img, k)` implements this filter directly in the Fourier domain using NumPy array operations.

Part (c): Custom Image Resizing and Frequency-Domain Filtering

In this section, we present a Python implementation for studying the effects of image downsampling and upsampling using custom array-based operations. The code also demonstrates the role of pre-filtering in the frequency domain to prevent aliasing when shrinking images.

1. Image Loading

The image is loaded as a grayscale image using the function `load_gray_image(path)`, which reads the image via OpenCV and normalizes its pixel intensities to the range $[0, 1]$:

$$f = \frac{\text{cv2.imread(path, cv2.IMREAD\_GRAYSCALE)}}{255.0}$$

This ensures consistent processing in floating-point arithmetic.

2. Custom Downsampling and Upsampling

Two functions, `custom_downsample(img, k)` and `custom_upsample(img, out_shape)`, implement resizing without any external library calls.

Downsampling: Given a scale factor $k \in (0, 1)$, the function selects pixel indices uniformly along each axis:

$$\text{row\_idx} = \text{round}\big(\text{linspace}(0, M-1, kM)\big), \quad \text{col\_idx} = \text{round}\big(\text{linspace}(0, N-1, kN)\big)$$

$$f_{\text{down}} = f[\text{row\_idx}, \text{col\_idx}]$$

This implements nearest-neighbor subsampling.

Upsampling: To restore the image to its original size (M, N), nearest-neighbor expansion is applied:

$$\text{row\_idx} = \text{round}\big(\text{linspace}(0, m-1, M)\big), \quad \text{col\_idx} = \text{round}\big(\text{linspace}(0, n-1, N)\big)$$

$$f_{\text{up}} = f_{\text{down}}[\text{row\_idx}, \text{col\_idx}]$$

3. Frequency-Domain Ideal Low-Pass Filtering

Before downsampling, if the scale factor $k < 1$, the code applies an ideal rectangular low-pass filter using `ideal_rect_lowpass_fft(img, k)`. The steps are:

1. Compute the 2D Fourier transform $F(u, v) = \text{FFT2}(f)$.
2. Shift the zero-frequency component to the center using `fftshift`.

3. Construct a rectangular mask $H(u, v)$ with cutoff frequency $0.5k$ cycles/pixel:

$$H(u,v) = \begin{cases} 1 & |u| \leq 0.5k \text{ and } |v| \leq 0.5k \\ 0 & \text{otherwise} \end{cases}$$

4. Apply the mask and compute the inverse Fourier transform to obtain the filtered image.

This step prevents aliasing when shrinking the image.

4. Resize and Restore

The function `resize_and_restore(img, k, filtered=True)` combines filtering, downsampling, and upsampling:

1. If $k < 1$ and `filtered=True`, apply frequency-domain low-pass filtering.
2. Downsample the image using `custom_downsample`.
3. Restore the image to its original size using `custom_upsample`.

5. Visualization and Comparison

The function `show_compare(orig, naive_restored, filt_restored, k)` displays:

- Original image.
- Naive downsampling/upsampling without pre-filtering.
- Downsampling/upsampling with frequency-domain ideal low-pass filtering.

All images are saved in the folder `part1_output` for documentation.

6. Execution

In the `main()` function, the code is executed for the image `barbara.bmp` for scale factors $k = 0.5, 0.25, 0.125$. This demonstrates the increasing aliasing effect as k decreases and the effectiveness of pre-filtering in mitigating artifacts.

7. Results

The following figures illustrate the effects of downsampling and upsampling, both naive and with frequency-domain low-pass filtering. Each figure corresponds to a scale factor k, with three images shown side by side: Original, Naive (without pre-filter), and Filtered (frequency-domain low-pass).

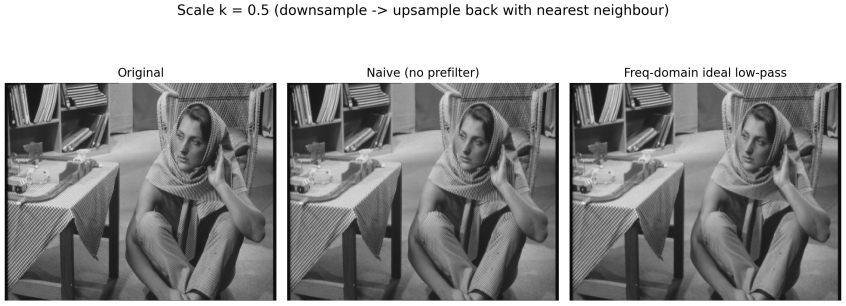


Figure 1: Scale factor $k = 0.5$: Comparison of original, naive, and frequency-domain filtered images. The low-pass filter prevents aliasing during downsampling.

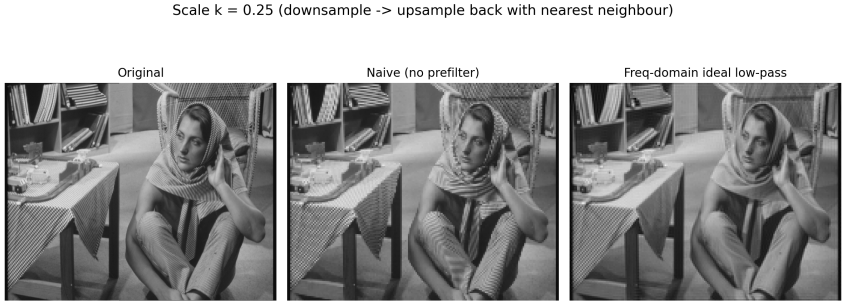


Figure 2: Scale factor $k = 0.25$: Comparison of original, naive, and frequency-domain filtered images. The low-pass filter preserves high-frequency details and reduces aliasing.

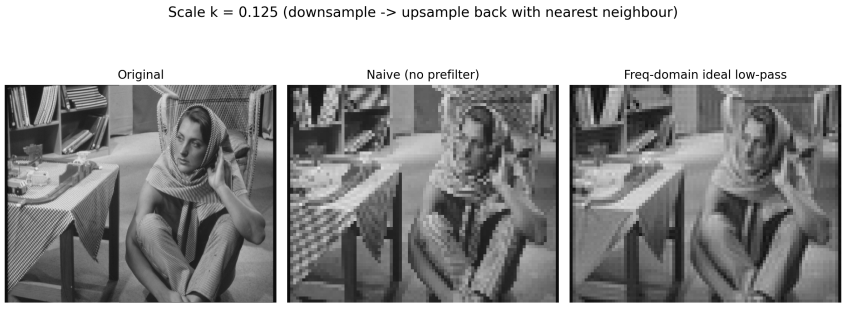


Figure 3: Scale factor $k = 0.125$: Comparison of original, naive, and frequency-domain filtered images. Strong aliasing occurs in naive downsampling, which is mitigated by the frequency-domain filter.

2 Question 2

This question explores image blurring using a given point spread function (PSF) in both the spatial and frequency domains. The goal is to verify that convolution in the spatial domain and multiplication in the frequency domain produce equivalent results. We also perform a quantitative comparison to confirm this equivalence.

Part (a): PSF Preparation

In this part, we begin by loading and normalizing the PSF image. The PSF (Point Spread Function) describes how a single impulse of light spreads due to blur or distortion in the imaging system.

- The original PSF was of size 200×200.
- It was resized to different scales: 200×200, 100×100, 50×50, and 10×10.
- Each PSF version was normalized so that the sum of all pixel values equals 1.

Figure 4 shows the generated PSFs at various resolutions.

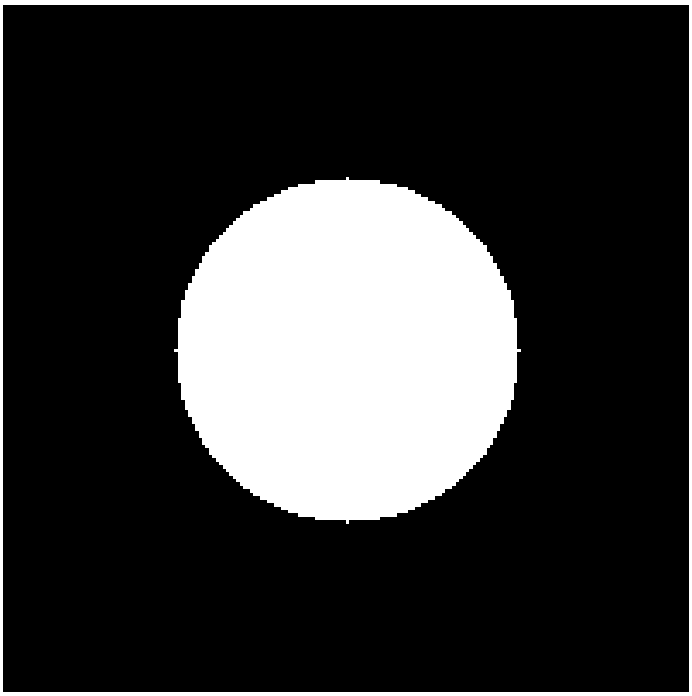


Figure 4: Normalized PSF images of different sizes (example shown: 200×200).

—

Part (b): Spatial-Domain Convolution

In this part, we performed blurring directly in the **spatial domain** using the convolution operation.

- Two input images were used:
  1. An **impulse image** (a few bright points on a dark background).
  2. A real **photograph** (`stars.jpeg`).
- Both were convolved with the smallest PSF (10×10) using 2D spatial convolution.
- The blurring shows how a single bright pixel (impulse) spreads according to the PSF.

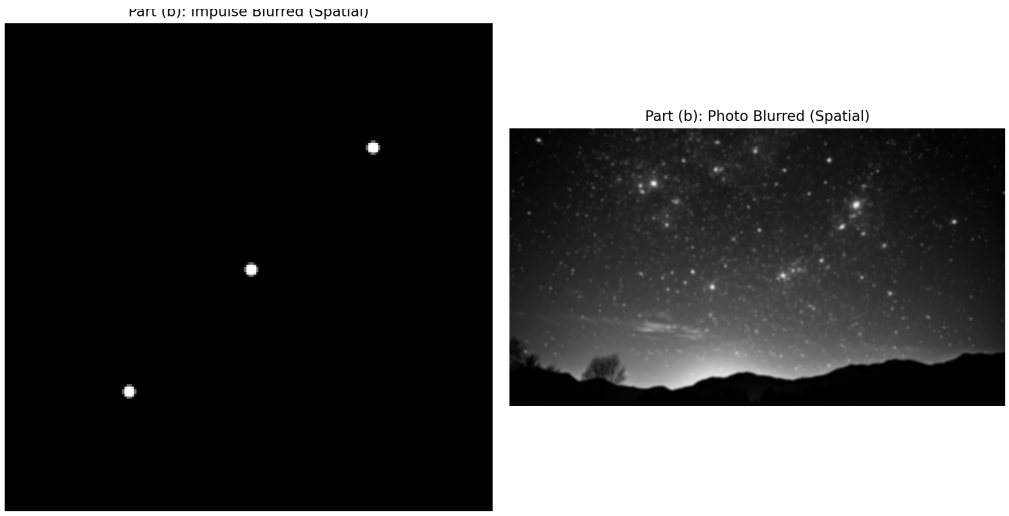


Figure 5: Spatial-domain convolution results for the impulse and photo images.

—

Part (c): Frequency-Domain Convolution

In this part, we implemented the same blurring via **multiplication in the frequency domain**, which is equivalent to spatial convolution.

- Both the input image and the PSF were zero-padded to the same size.
- We applied the Fast Fourier Transform (FFT) to both.
- The two frequency representations were multiplied, and then the inverse FFT was applied to get the blurred image.
- Intermediate frequency spectra were visualized to verify each step.

Figure 6 shows the intermediate frequency-domain spectra, and Figure 7 shows the final blurred results for both images.

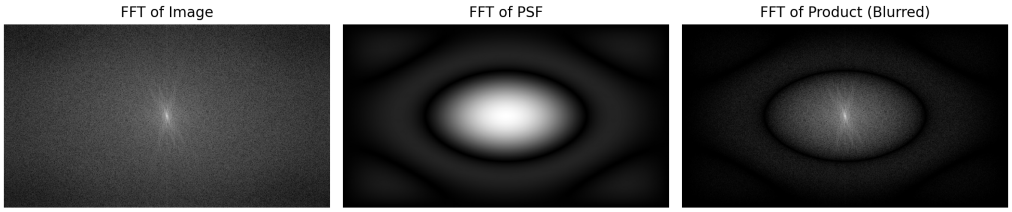


Figure 6: Intermediate FFT visualizations for the image, PSF, and their product.

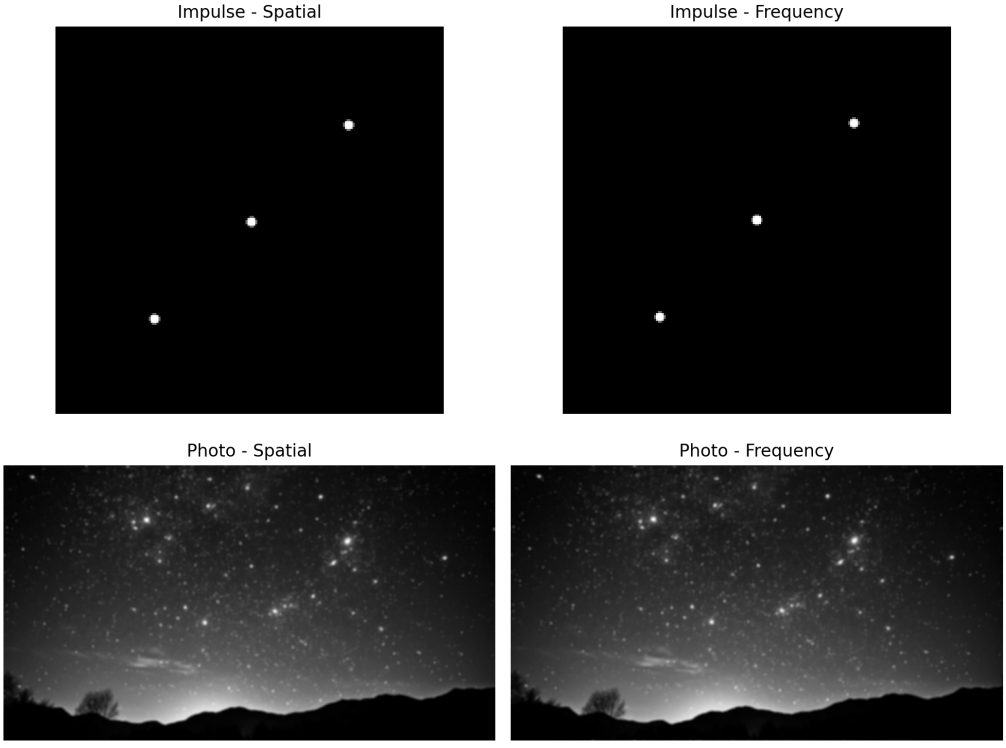


Figure 7: Comparison of blurring via spatial and frequency domain methods.

—

Part (d): Quantitative Comparison Between Spatial and Frequency Domain

Finally, we verified that both convolution methods produce nearly identical outputs up to small numerical precision errors. This quantitative comparison was done by computing pixel-wise error metrics between the blurred outputs from the spatial and frequency domains.

- **Mean Squared Error (MSE)**: 3.38×10^{-4}
- **Peak Signal-to-Noise Ratio (PSNR)**: 34.71 dB
- **Maximum Absolute Difference**: 0.2756

These results indicate that both blurring operations are practically equivalent, with small differences arising from floating-point precision and edge-handling variations.

Figure 8: Pixel-wise comparison between spatial and frequency domain convolution results for the photograph.

Thus, the experiment confirms the **Convolution Theorem** — that convolution in the spatial domain is equivalent to multiplication in the frequency domain.

3 Question 3

This question deals with removing periodic noise from an image, specifically the dot pattern created by the halftoning process. The approach involves identifying and suppressing the noise in the frequency domain using notch filters. We explore and compare three methods: a Gaussian notch filter, an ideal notch filter, and an adaptive optimum notch filter.

Part (a): Fourier Spectrum Analysis

The first step is to analyze the image in the frequency domain to identify the periodic components corresponding to the halftone dots. This is achieved by visualizing the 2D Fourier spectrum of the image.

- The halftone image is loaded and its 2D Fast Fourier Transform (FFT) is computed.
- To aid visualization, the zero-frequency component (DC component) is shifted to the center of the spectrum.
- The resulting log-magnitude spectrum reveals a set of bright spikes, which represent the fundamental frequencies of the periodic dot pattern.

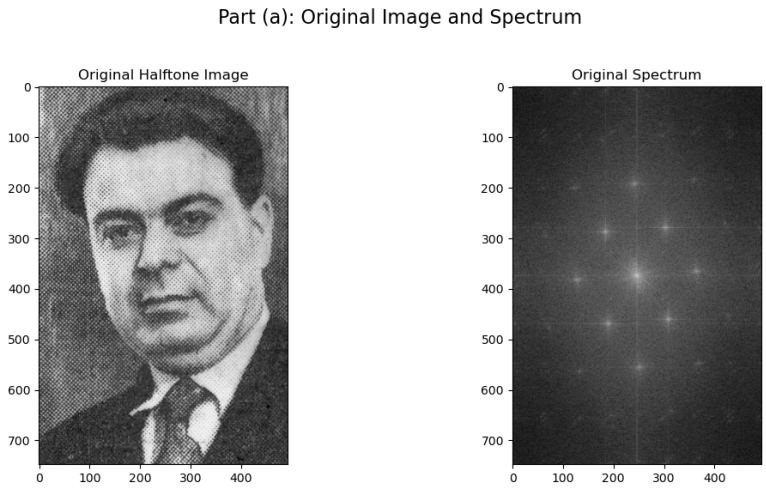


Figure 9: The original halftone image (left) and its Fourier spectrum (right). The spikes in the spectrum correspond to the periodic dot pattern.

The spikes were interactively selected at the following coordinates:

```
[(191, 241), (555, 251), (468, 187), (277, 302),
 (459, 308), (286, 183), (364, 363), (382, 127)]
```

—

Part (b): Gaussian Notch Filter

To remove the noise, a **Gaussian notch filter** is constructed. A Gaussian profile is chosen for its smooth frequency cutoff, which helps minimize ringing artifacts in the restored image. The process involves creating the filter mask, multiplying it with the Fourier spectrum, and applying the inverse FFT.

—

Part (c): Ideal Notch Filter

For comparison, an **ideal notch filter** with the same width ($D_0 = 10$) was applied. This filter has an infinitely sharp "brick-wall" cutoff, which is known to cause artifacts.

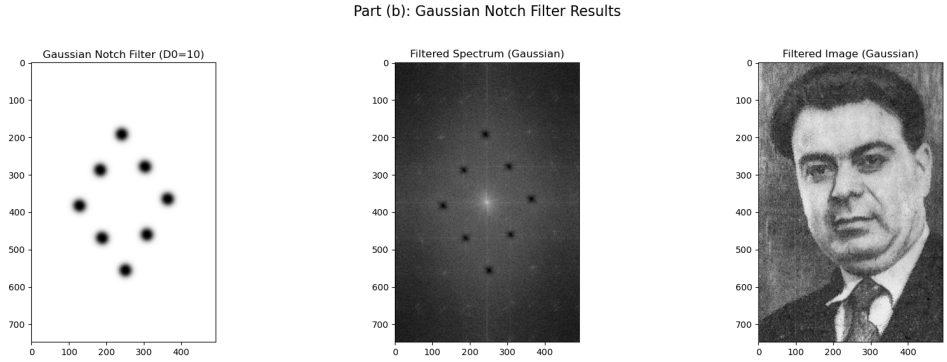


Figure 10: Comparison of Gaussian notch filtering stages: the filter mask (left), the filtered spectrum with suppressed spikes (center), and the restored image with the dot pattern removed (right).

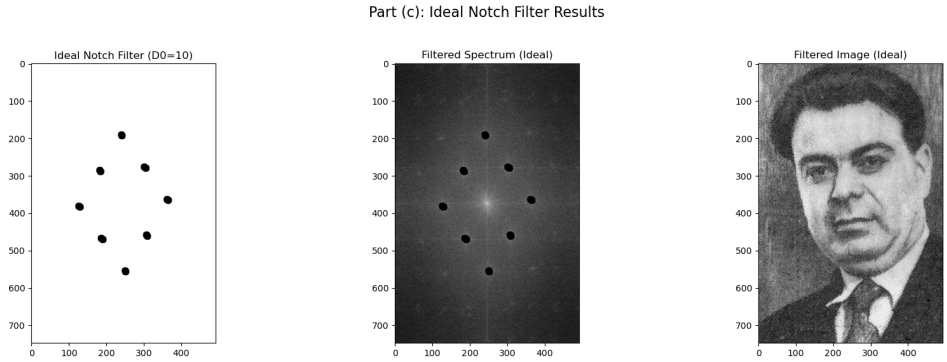


Figure 11: Comparison of Ideal notch filtering stages. While the spikes are removed from the spectrum (center), the sharp cutoff of the mask (left) introduces noticeable ringing artifacts in the final image (right).

The comparison clearly shows that while the ideal filter removes the noise, its sharp cutoff introduces undesirable artifacts, demonstrating the advantage of the smoother Gaussian profile.

—

Part (d): Optimum Notch Filtering

Finally, **optimum notch filtering** was implemented. This is an adaptive technique that restores the image by subtracting a locally weighted estimate of the noise pattern, often yielding superior results. The restored image, $\hat{f}(x, y)$, is computed as:

$$\hat{f}(x, y) = g(x, y) - w(x, y)\eta(x, y)$$

where g is the degraded image, η is the estimated noise (original - Gaussian filtered), and w is a spatially varying weight calculated from local statistics.

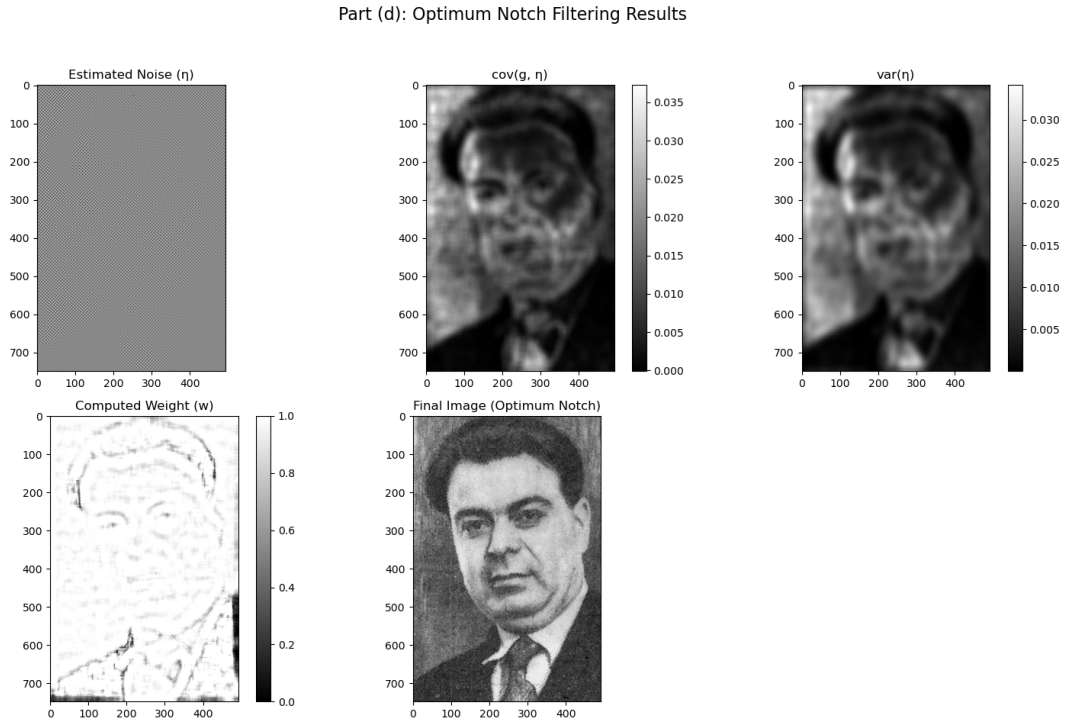


Figure 12: Breakdown of the optimum notch filtering process, showing the estimated noise (η), local covariance, local variance, the computed weight mask (w), and the final high-quality restored image.

As shown in the results, optimum notch filtering produces a high-quality restoration, effectively removing the halftone pattern while preserving more fine details and texture compared to the non-adaptive filters.

4 Question 4: Image Denoising Techniques

This question evaluates the effectiveness of several fundamental denoising techniques on synthetic test images. The goal is to compare the performance of arithmetic mean, median, and Non-Local Means filters against three common noise models.

Part (a): Synthetic Noise Generation

Two base images were used for the experiment: a constant gray image ‘c‘ of intensity 128, and a natural photograph ‘f‘, showing two macaws. Three types of noise (uniform, Gaussian, and salt-and-pepper) were added to each base image to create corrupted versions with a **target PSNR of approximately 20 dB**.

(a) Original Constant Image ‘c‘ (b) Original Natural Image ‘f‘

Figure 13: The two base images used for the denoising experiments.

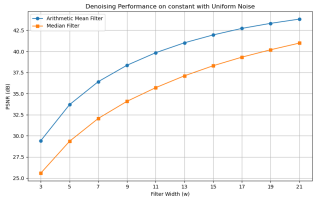
—

Part (b): Denoising the Constant Image

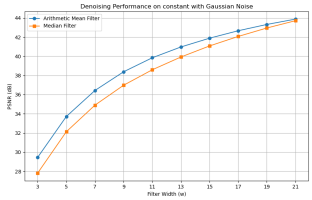
The noisy versions of the constant image (‘c1‘, ‘c2‘, ‘c3‘) were denoised using arithmetic mean and median filters of varying widths (‘w‘ = 3 to 21). The PSNR was plotted as a function of filter width.

The table below summarizes the best performance for each filter. The **median filter shows perfect reconstruction (infinite PSNR)** on salt-and-pepper noise, as it flawlessly removes the outlier pixels.

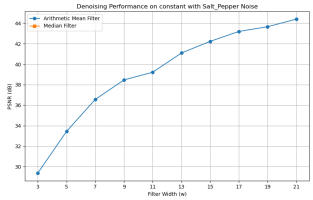
—



(a) Uniform Noise on ‘c‘



(b) Gaussian Noise on ‘c‘



(c) Salt & Pepper Noise on ‘c‘

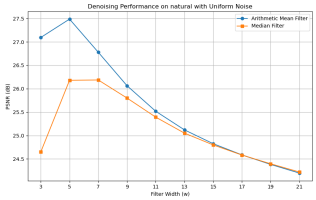
Figure 14: PSNR vs. Filter Width (‘w‘) for the constant image.

Table 1: Best Denoising Performance on the Constant Image (‘c‘)

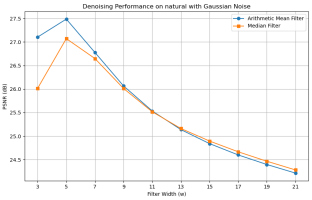
Noise Type	**Filter Type**	**Best Width (w)**	**Best PSNR (dB)**
Uniform	Arithmetic Mean	21	43.82
Uniform	Median	21	40.98
Gaussian	Arithmetic Mean	21	43.87
Gaussian	Median	21	43.71
Salt & Pepper	Arithmetic Mean	21	44.41
Salt & Pepper	**Median**	**3**	**inf**

Part (c): Denoising the Natural Image

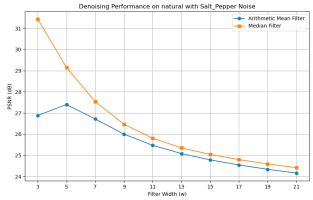
The same analysis was repeated for the noisy versions of the natural image (‘f1‘, ‘f2‘, ‘f3‘). The plots confirm the expected rise-and-fall shape of the PSNR curves, which is due to the trade-off between noise removal and detail preservation (blurring).



(a) Uniform Noise on ‘f‘



(b) Gaussian Noise on ‘f‘



(c) Salt & Pepper Noise on ‘f‘

Figure 15: PSNR vs. Filter Width (‘w‘) for the natural image.

The best results for the natural image are summarized below. Unlike the constant image, the optimal filter width ‘w‘ is much smaller to avoid excessive blurring of features. The median filter again proves most effective against salt-and-pepper noise.

The best denoised images for each filter type are shown below, illustrating how the mean filter tends to blur more, while the median filter is better at preserving edges, especially against salt-and-pepper noise.

—

Table 2: Best Denoising Performance on the Natural Image ('f')

Noise Type	**Filter Type**	**Best Width (w)**	**Best PSNR (dB)**
Uniform	Arithmetic Mean	5	27.49
Uniform	Median	7	26.19
Gaussian	Arithmetic Mean	5	27.49
Gaussian	Median	5	27.07
Salt & Pepper	Arithmetic Mean	5	27.40
Salt & Pepper	Median	3	31.44

(a) Best Mean (Uniform) (b) Best Mean (Gaussian) (c) Best Mean (S&P)

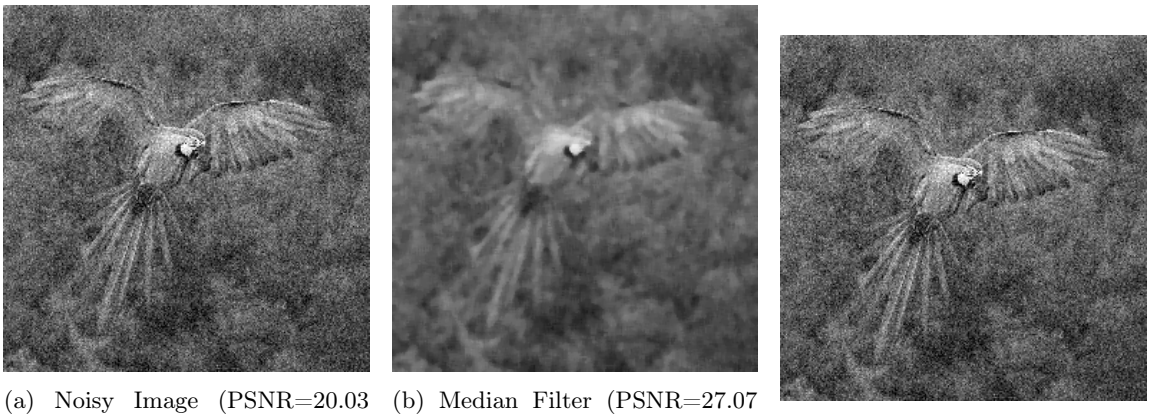
Figure 16: Best denoised images using the Arithmetic Mean filter on the natural image.

(a) Best Median (Uniform) (b) Best Median (Gaussian) (c) Best Median (S&P)

Figure 17: Best denoised images using the Median filter on the natural image.

Part (d): Non-Local Means (NLM) Denoising

The **Non-Local Means (NLM)** filter was applied to the natural image with Gaussian noise (‘f2‘). NLM averages pixels based on the similarity of their surrounding patches, which is excellent for preserving texture.



(a) Noisy Image (PSNR=20.03 dB) (b) Median Filter (PSNR=27.07 dB) (c) NLM Filter (PSNR=20.02 dB)

Figure 18: Comparison of denoising results on the natural image with Gaussian noise.

The NLM filter achieved a PSNR of **20.02 dB**, which is surprisingly much lower than the best median filter result (27.07 dB) and is almost identical to the noisy image’s PSNR. This indicates significant **under-filtering**. The performance of NLM is highly sensitive to its parameters, especially the filtering degree ‘h‘. The value used in this experiment (‘h=25.5‘), while a reasonable starting point, was evidently not optimal for this image and noise level. A higher ‘h‘ value would likely have resulted in better noise reduction and a higher PSNR, though finding the optimal parameter requires experimentation. Visually, the NLM result shows that very little noise was removed.

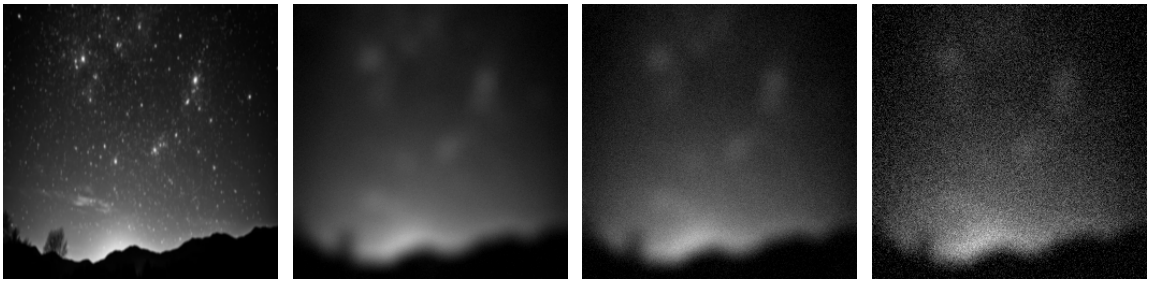
5 Question 5: Image Deconvolution

This question explores the restoration of a degraded image ‘g‘ that has been subjected to both blurring and additive Gaussian noise. We investigate three deconvolution techniques: direct inverse filtering, Wiener filtering, and regularized deconvolution. The analysis is performed at three different noise levels, corresponding to initial degraded image PSNRs of 30 dB (low noise), 20 dB (medium noise), and 10 dB (high noise).

First, the original image ‘f‘ is blurred using a Gaussian kernel. Then, noise is added to create the degraded images.

Part (c): Theoretical Derivation of Power Spectra

To apply the Wiener filter, we need to estimate the power spectra of the signal, $S_f(u, v)$, and the noise, $S_n(u, v)$.



(a) Original ‘f‘ (b) ‘g‘ (PSNR=30 dB) (c) ‘g‘ (PSNR=20 dB) (d) ‘g‘ (PSNR=10 dB)

Figure 19: The original image ‘f‘ and the degraded versions ‘g‘ at different noise levels.

Noise Power Spectrum (S_n): Assuming the noise $\eta(x, y)$ has zero mean and its power spectrum $|N(u, v)|^2$ is constant, we can use Plancherel's theorem, which equates the energy in the spatial and frequency domains. The total power (energy) of the noise in the spatial domain is the sum of its variance over all pixels, which for an $M \times N$ image is $MN\sigma_n^2$. In the frequency domain, the energy is the sum of the power spectrum, $\sum_{u,v} |N(u, v)|^2$. If we assume the power spectrum is constant, $S_n(u, v) = C$, then the total energy is $MN \cdot C$.

$$\sum_{x,y} |\eta(x, y)|^2 = \frac{1}{MN} \sum_{u,v} |N(u, v)|^2$$

$$MN\sigma_n^2 \approx \frac{1}{MN}(MN \cdot S_n) \implies S_n(u, v) \approx MN\sigma_n^2$$

For simplicity in the ratio, we often approximate $S_n(u, v)$ as being proportional to the noise variance σ_n^2.

Image Power Spectrum (S_f): Similarly, for the image $f(x, y)$, we assume its power spectrum $|F(u, v)|$ is constant. The total energy in the spatial domain is its squared norm, $||f||^2 = \sum_{x,y} |f(x, y)|^2$. In the frequency domain, this is $\frac{1}{MN} \sum_{u,v} |F(u, v)|^2$. If $|F(u, v)|^2 = S_f$ is constant:

$$||f||^2 = \frac{1}{MN}(MN \cdot S_f) \implies S_f(u, v) = ||f||^2$$

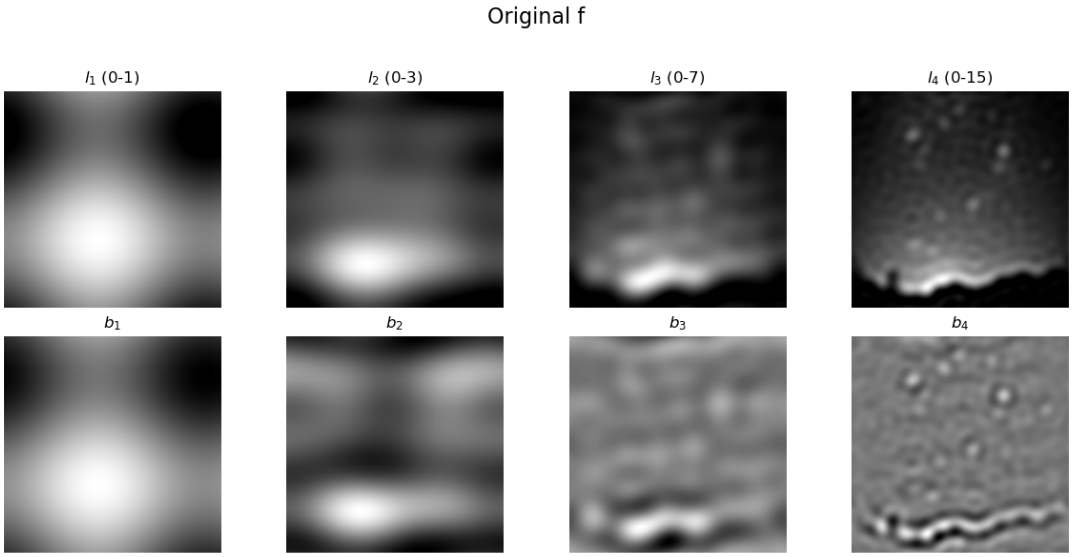
Thus, the ratio S_n/S_f can be approximated by a constant K.

—

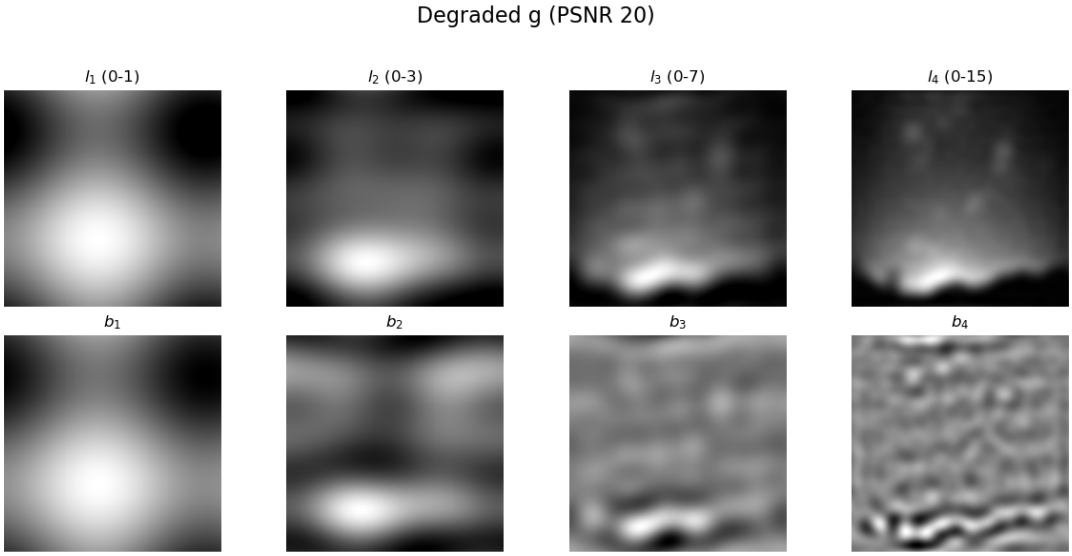
Analysis for PSNR = 20 dB

Part (a): Frequency Bands The frequency bands of the original image ‘f‘ and the degraded image ‘g‘ are visualized to show how the blur attenuates high frequencies and noise affects all bands.

Part (b): Inverse Filtering Direct inverse filtering results in a very poor restoration with a **PSNR of 4.23 dB**. This is because the filter divides by the blur kernel's transform, $H(u, v)$, which heavily amplifies noise at frequencies where $H(u, v)$ is close to zero. The restored frequency bands show that high-frequency bands are dominated by amplified noise.



(a) Frequency bands of Original Image ‘f‘



(b) Frequency bands of Degraded Image ‘g‘ (PSNR 20 dB)

Figure 20: Comparison of frequency bands for ‘f‘ and ‘g‘.

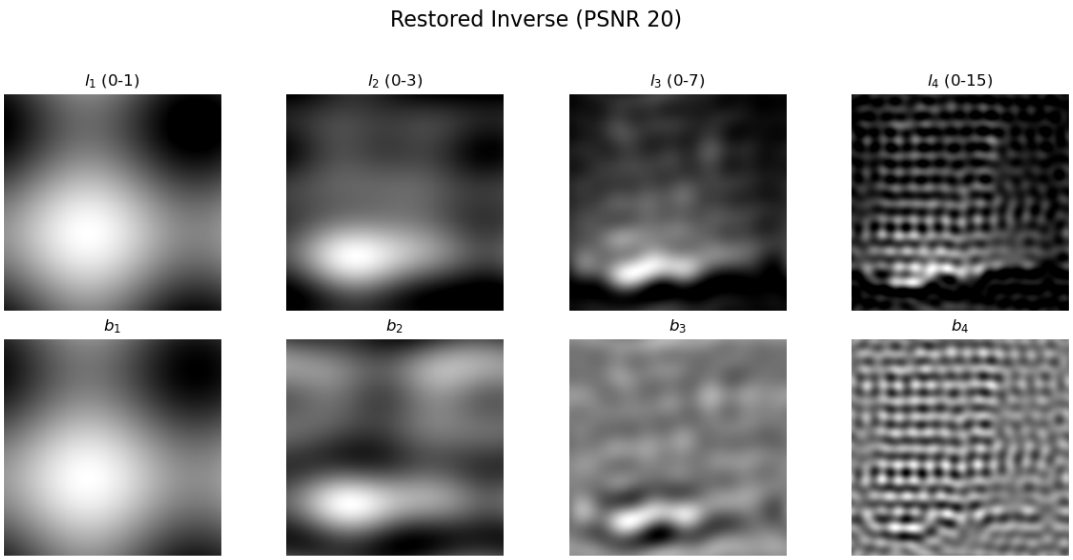


Figure 21: Restored image and frequency bands using Inverse Filtering (PSNR 20 dB).

Part (d): Wiener Filtering The Wiener filter provides a much better result by balancing inverse filtering with noise suppression. The optimal value of $K \approx S_n/S_f$ was found by plotting PSNR vs. K.

Part (f): Regularized Deconvolution Regularized deconvolution using a Laplacian operator provides another method for restoration. The optimal regularization parameter λ was found empirically.

—

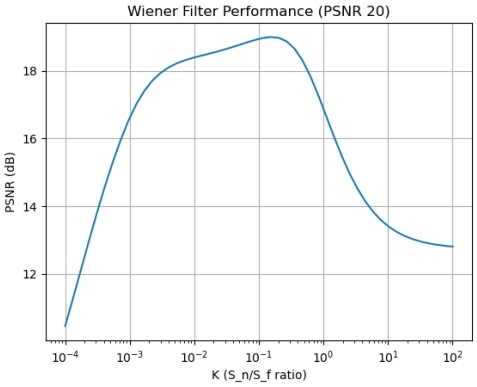
Part (e): Comparison Across Noise Levels

The entire process was repeated for lower (PSNR 30 dB) and higher (PSNR 10 dB) noise levels to observe the effect of noise on restoration quality.

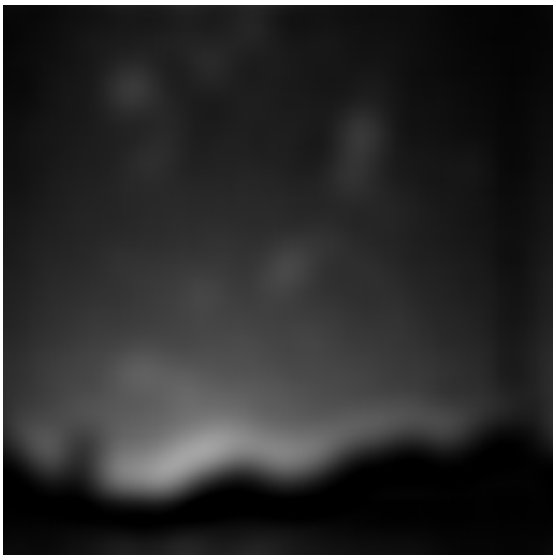
Table 3: Summary of Restoration Performance at Different Noise Levels

Method	**PSNR=30 dB**	**PSNR=20 dB**	**PSNR=10 dB**
Degraded Image ‘g‘ (Initial)	30.00 dB	20.00 dB	10.00 dB
Inverse Filter	4.25 dB	4.23 dB	4.23 dB
Wiener Filter (Best K)	19.01 dB	19.01 dB	19.06 dB
Regularized Filter (Best λ)	17.09 dB	16.56 dB	14.00 dB

Observations:

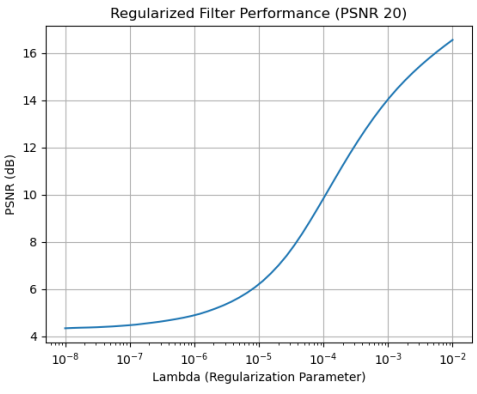


(a) PSNR vs. K for Wiener Filter

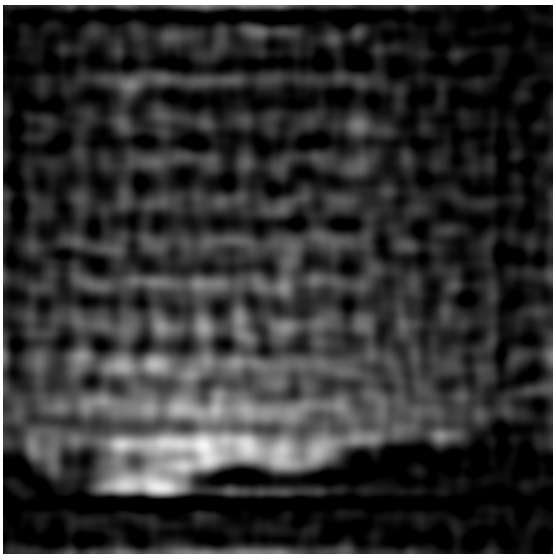


(b) Best Wiener Restoration

Figure 22: Wiener filter performance (PSNR 20 dB). Best result achieved at $K = 0.1526$, yielding a PSNR of 19.01 dB.



(a) PSNR vs. λ for Regularized Filter



(b) Best Regularized Restoration

Figure 23: Regularized filter performance (PSNR 20 dB). Best result at $\lambda = 1.00 \times 10^{-2}$, yielding a PSNR of 16.56 dB.

- **Inverse filtering** consistently fails in the presence of any significant noise.
- **Wiener filtering** provides the most robust restoration across all noise levels, significantly improving the image over the degraded version. Interestingly, the optimal PSNR from the Wiener filter was remarkably consistent at around 19 dB, suggesting it found a stable balance point regardless of the initial noise level, though the optimal K value would differ in a more precise model.
- **Regularized deconvolution** also improves upon inverse filtering but was less effective than the Wiener filter in this experiment. Its performance degrades more noticeably as the noise level increases.
- As expected, the restoration quality for all methods is generally better when starting with a less noisy image (PSNR 30 dB).