In [2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def add_gaussian_noise(img, sigma):
    # Convert to float for safe addition
    noisy = img.astype(np.float64)
    # Generate Gaussian noise (mean = 0, std = sigma)
    noise = np.random.normal(0, sigma, img.shape)
    # Add noise
    noisy += noise
    # Clip to valid range (0-255 for 8-bit images)
    noisy = np.clip(noisy, 0, 255)
    return noisy.astype(np.uint8)


img = cv2.imread('taj_clean.jpg', cv2.IMREAD_GRAYSCALE)
noisy_img = add_gaussian_noise(img, sigma=20)
cv2.imwrite('taj_noisy.png', noisy_img)
```

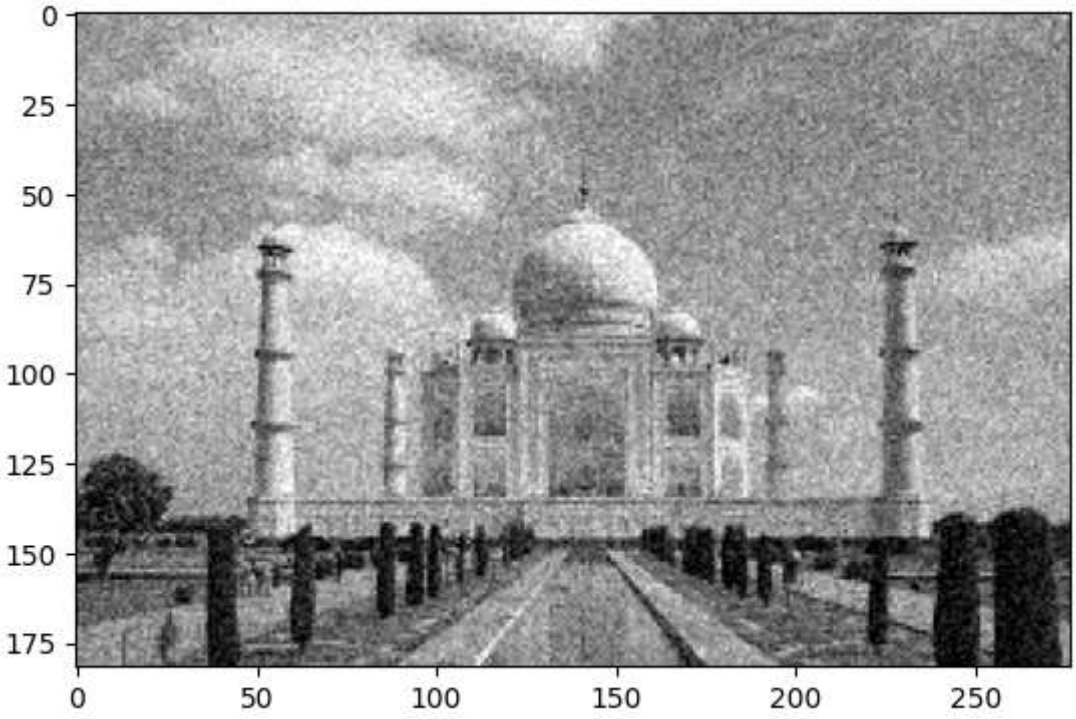
Out[2]:
```
True
```

In [3]:
```
def process_pixel(img, i, j, k=9, b=25.0):
    r = k//2
    padded = np.pad(img, r, mode='constant', constant_values=0)
    pi, pj = i + r, j + r
    K1 = padded[pi-r:pi+r+1, pj-r:pj+r+1].astype(np.float64)
    mew = padded[pi-1:pi+1+1, pj-1:pj+1+1].astype(np.float64)
    mew = (1/9)*(mew.sum())
    a = padded[pi, pj]
    a = mew
    K2 = np.exp(-((K1 - a)**2) / (2.0 * (b**2)))
    mn = K2.min()
    mx = K2.max()
    K3 = (K2 - mn) / (mx - mn + 1e-12)
    K4 = (K3 >= 0.8).astype(np.float64)
    wsum = K4.sum()
    if wsum == 0:
        return float(a)
    patch = K1
    return float((K4 * patch).sum() / wsum)

def reconstruct_image(img, k=9, b=25.0):
    img_arr = img.astype(np.float64)
    H, W = img_arr.shape
    out = np.zeros_like(img_arr)
    for i in range(H):
        for j in range(W):
            out[i, j] = process_pixel(img_arr, i, j, k=k, b=b)
    return out
```

In [4]:
```
im2 = reconstruct_image(noisy_img,9,50)
bilateral = cv2.bilateralFilter(noisy_img, 9, 50, 50)
```

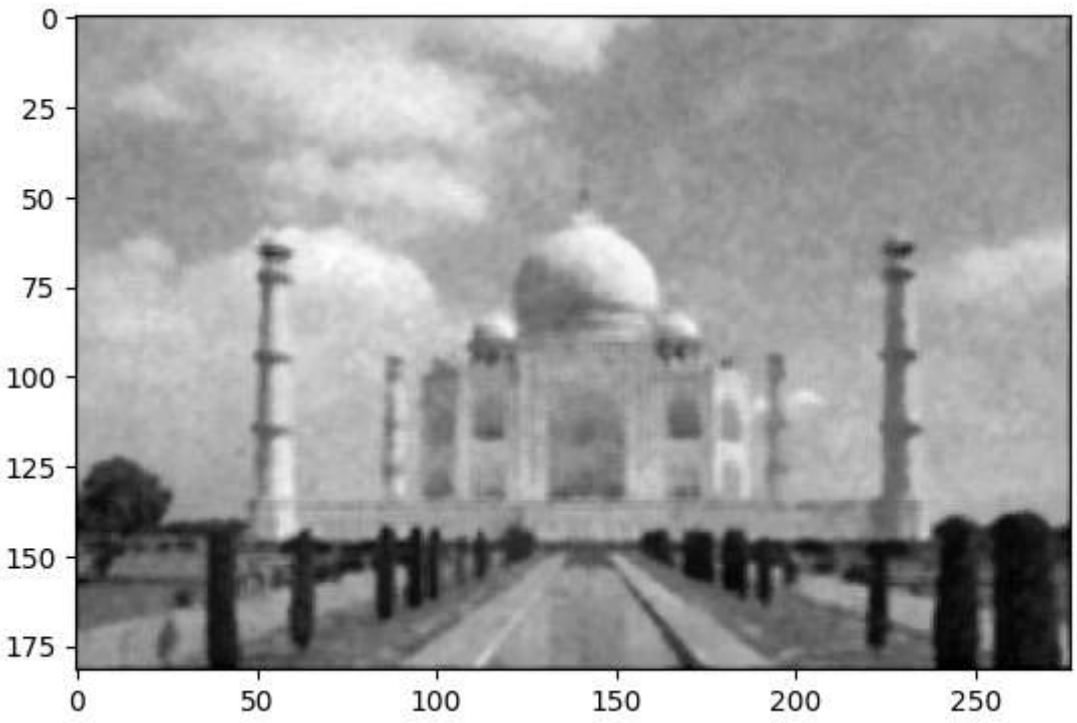
In [136]:
```
plt.imshow(noisy_img, cmap = "gray")
```

Out[136]:
```
<matplotlib.image.AxesImage at 0x1379b1a2170>
```



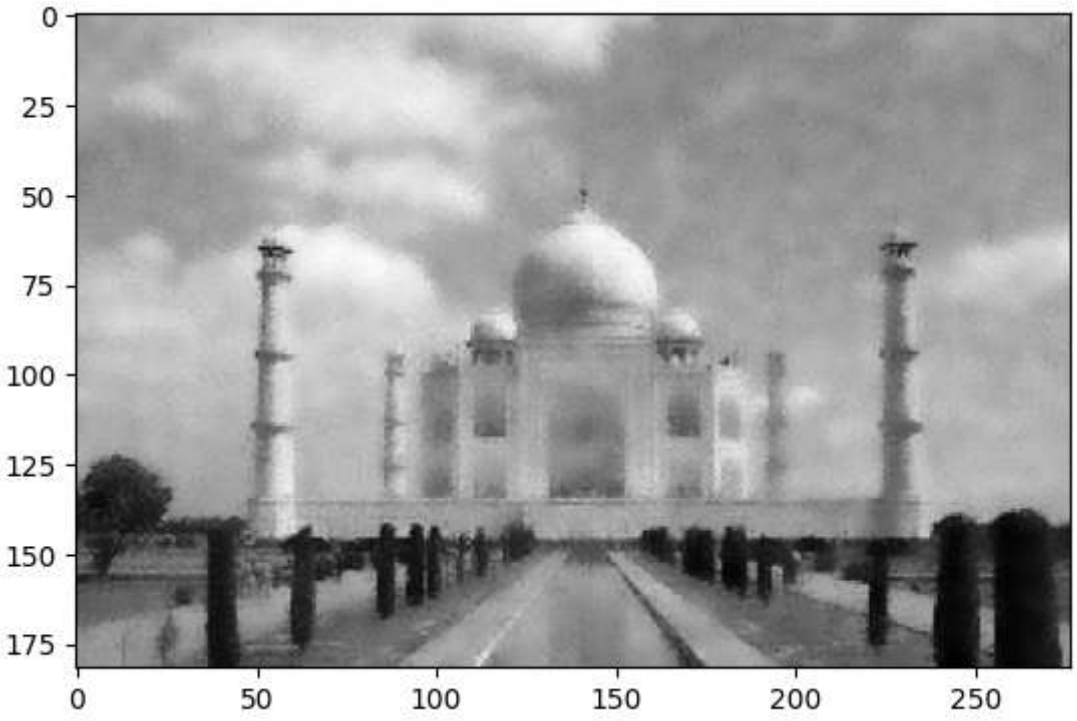
In [137]:
```
plt.imshow(im2,cmap = "gray")
```

Out[137]:
```
<matplotlib.image.AxesImage at 0x1379b2740a0>
```



In [138]:
```
plt.imshow(bilateral,cmap = "gray")
```

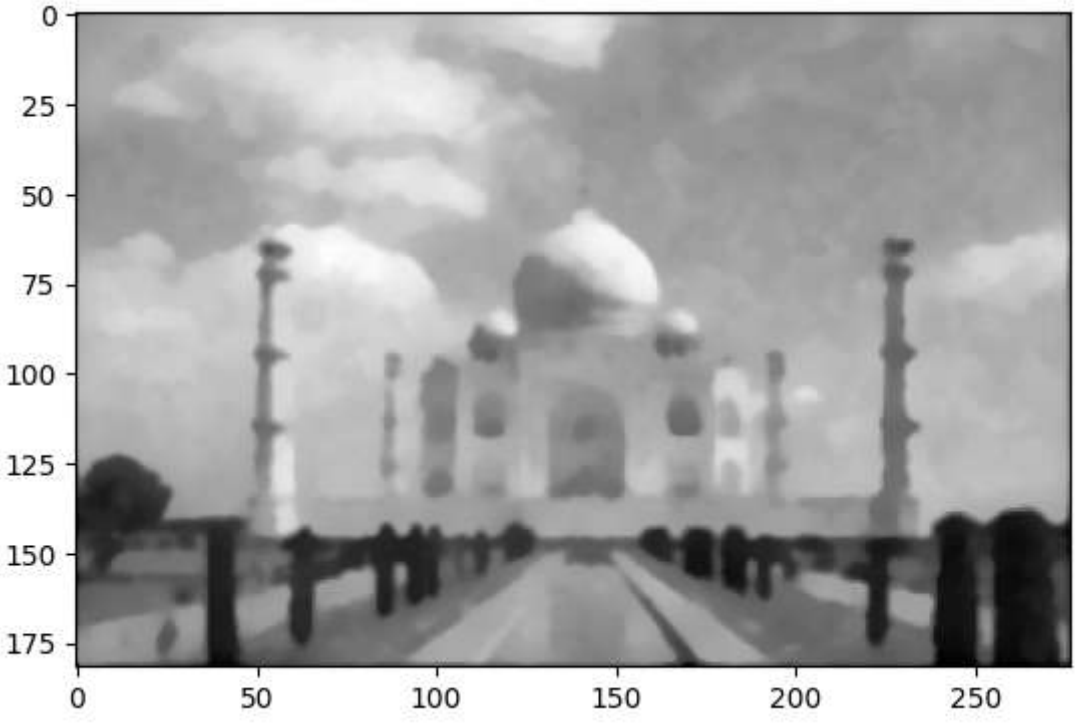
Out[138]:
```
<matplotlib.image.AxesImage at 0x1379b2caef0>
```



In [139]:
```
im3 = reconstruct_image(im2,9,50)
```

In [140]:
```
plt.imshow(im3,cmap = "gray")
```

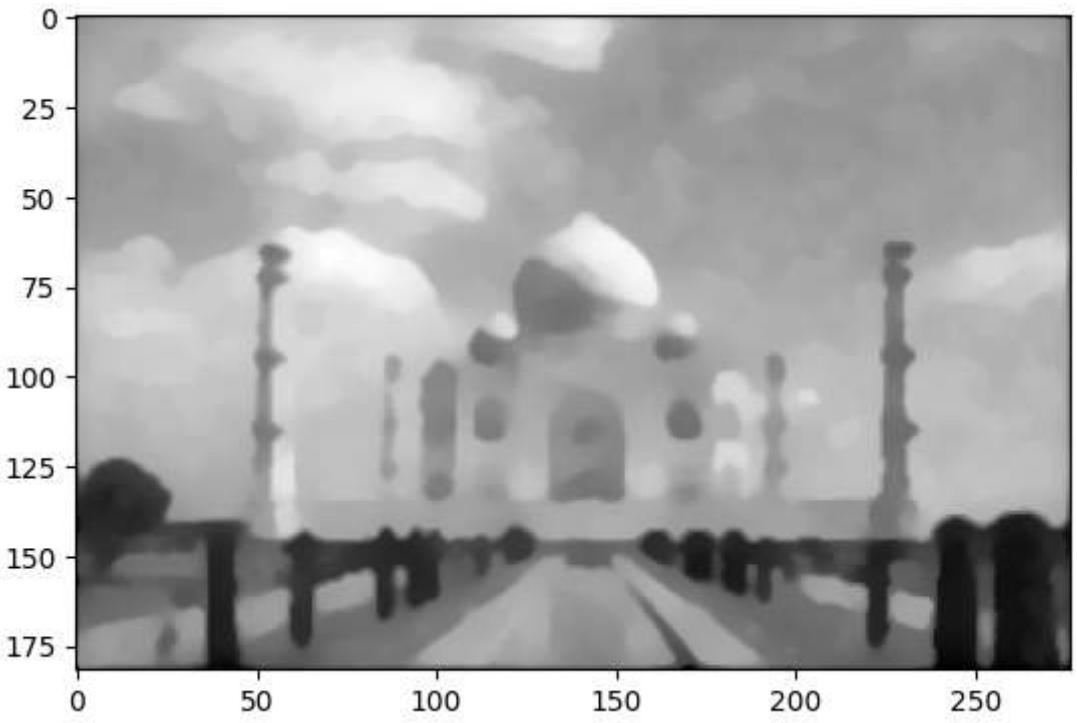
Out[140]:
```
<matplotlib.image.AxesImage at 0x1379b486ad0>
```



In [141]:
```
im4 = reconstruct_image(im3,9,50)
```

In [142]:
```
plt.imshow(im4,cmap = "gray")
```

Out[142]:
```
<matplotlib.image.AxesImage at 0x1379b505d50>
```



In [143]:
```
from skimage.metrics import peak_signal_noise_ratio as psnr, structural_similarit
def eval_pair(clean, recon):
    return psnr(clean, recon, data_range=clean.max()-clean.min()), \
           ssim(clean, recon, data_range=clean.max()-clean.min())
```

In [144]:
```
eval_pair(img, bilateral)
```

Out[144]:
```
(np.float64(27.131354194557602), np.float64(0.7970509452408788))
```

In [145]:
```
eval_pair(img, im2)
```

Out[145]:
```
(np.float64(25.315308131642542), np.float64(0.7456215668798571))
```

In [146]:
```
eval_pair(img, im3)
```

Out[146]:
```
(np.float64(24.652714856102737), np.float64(0.7626621825217434))
```

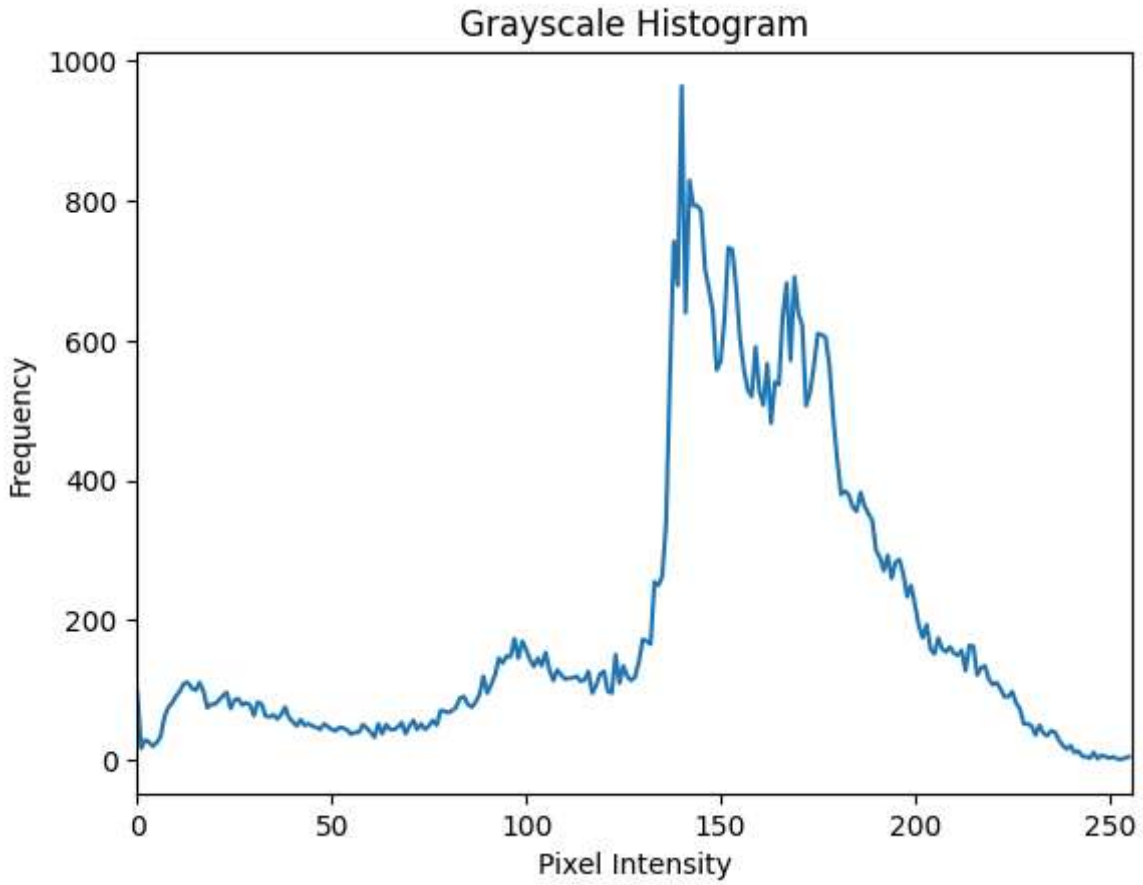
In [147]:
```
eval_pair(img, im4)
```

Out[147]:
```
(np.float64(23.82101218419555), np.float64(0.7324692693877091))
```

In [148]:
```
np.max(im4),np.min(im4)
```

Out[148]:
```
(np.float64(235.1217602267215), np.float64(0.032317636195752536))
```

In [149]:
```
import cv2
import matplotlib.pyplot as plt

# Load the image in grayscale
#image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Compute histogram
hist = cv2.calcHist([np.array(img,dtype=np.uint8)], [0], None, [256], [0, 256])

# Plot the histogram
plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.plot(hist)
plt.xlim([0, 256])
plt.show()
```



In [150]:
```
np.shape(img)
```

Out[150]:
```
(182, 277)
```

In [151]:
```
np.shape(im4)
```

Out[151]:
```
(182, 277)
```

In [152]:
```
eval_pair(img, img+1)
```

Out[152]:
```
(np.float64(39.40982254755878), np.float64(0.9977000759296464))
```

In [153]:
```
import numpy as np

def cosine_similarity(img1, img2):
    # Flatten both images into 1D vectors
    v1 = img1.flatten().astype(np.float64)
    v2 = img2.flatten().astype(np.float64)

    # Subtract mean to remove brightness bias (optional but recommended)
    v1 -= v1.mean()
    v2 -= v2.mean()

    # Compute cosine similarity
    numerator = np.dot(v1, v2)
    denominator = np.linalg.norm(v1) * np.linalg.norm(v2)

    if denominator == 0:
        return 0.0
    return numerator / denominator
```

In [154]:
```
cosine_similarity(img, bilateral)
```

Out[154]:
```
np.float64(0.9724141895242868)
```

In [9]:
```
import numpy as np

def image_covariance(img1, img2):
    # Flatten both images
    v1 = img1.flatten().astype(np.float64)
    v2 = img2.flatten().astype(np.float64)

    # Subtract means
    v1 -= v1.mean()
    v2 -= v2.mean()

    # Compute covariance
    cov = np.mean(v1 * v2)
    return cov
```

In [156]:
```
image_covariance(img, bilateral)/ (img.std() * bilateral.std())
```

Out[156]:
```
np.float64(0.9724141895242869)
```

In [157]:
```
image_covariance(img, im2)/ (img.std() * im2.std())
```

Out[157]:
```
np.float64(0.9569573737074497)
```

In [158]:
```
image_covariance(img, noisy_img)/ (img.std() * noisy_img.std())
```

Out[158]:
```
np.float64(0.9212735556044664)
```

In [ ]:
```

```

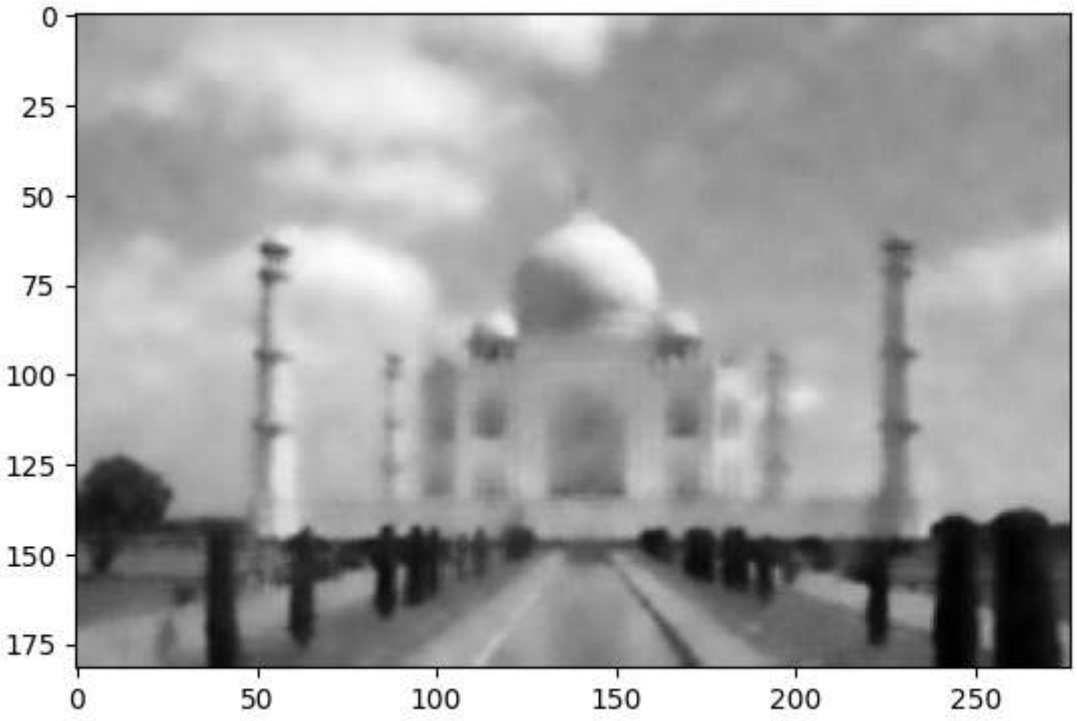
In [1]:
```
import numpy as np

def modified_bilateral(img, d=7, sigma_spatial=3.0, sigma_intensity=25.0):
    """
    Modified bilateral filter:
      - img: 2D grayscale image (numpy array, dtype uint8 or float)
      - d: kernel size (odd, e.g. 5 or 7)
      - sigma_spatial: spatial Gaussian sigma
      - sigma_intensity: intensity-domain sigma applied to (mean_p - q)
        where mean_p is the mean of the 3x3 neighborhood centered at p.
    Returns filtered image (same dtype as input).
    """
    # ensure float
    in_dtype = img.dtype
    I = img.astype(np.float64)
    H, W = I.shape
    r = d // 2

    # compute mean_p = mean of 3x3 neighborhood (including center)
    try:
        # fast path: use OpenCV if available
        import cv2
        mean_p = cv2.blur(I.astype(np.float32), ksize=(3,3)).astype(np.float64)
    except Exception:
        # fallback: sliding window average for 3x3
        pad = np.pad(I, 1, mode='reflect')
        mean_p = np.zeros_like(I)
        for i in range(H):
            for j in range(W):
                mean_p[i,j] = pad[i:i+3, j:j+3].mean()

    # pad image for patch extraction
    padded = np.pad(I, r, mode='reflect')

    # spatial kernel
    ax = np.arange(-r, r+1)
    xx, yy = np.meshgrid(ax, ax)
    S = np.exp(-(xx**2 + yy**2) / (2.0 * (sigma_spatial**2)))
    S = S / (S.sum() + 1e-12)

    # use numpy sliding windows if available for vectorized ops
    try:
        from numpy.lib.stride_tricks import sliding_window_view
        patches = sliding_window_view(padded, (d, d))  # shape (H, W, d, d)
        # compute range weights based on mean_p
        mp = mean_p[:, :, None, None]  # shape (H, W, 1, 1)
        diff = patches - mp  # broadcast
        R = np.exp(-(diff**2) / (2.0 * (sigma_intensity**2)))
        W = R * S[None, None, :, :]  # combine with spatial kernel
        W_sum = W.sum(axis=(2,3), keepdims=True)
        W_norm = W / (W_sum + 1e-12)
        out = (W_norm * patches).sum(axis=(2,3))
    except Exception:
        # fallback slower loop
        out = np.zeros_like(I)
        for i in range(H):
            for j in range(W):
```

```
                pi = i + r
                pj = j + r
                patch = padded[pi-r:pi+r+1, pj-r:pj+r+1]
                mp = mean_p[i, j]
                R = np.exp(-((patch - mp)**2) / (2.0 * (sigma_intensity**2)))
                W = R * S
                W = W / (W.sum() + 1e-12)
                out[i, j] = (W * patch).sum()

    # cast back to original dtype/range
    if np.issubdtype(in_dtype, np.integer):
        out = np.clip(out, 0, 255)
        return out.astype(in_dtype)
    else:
        return out.astype(in_dtype)

# Example usage:
# import cv2
# img = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
# filtered = modified_bilateral(img, d=7, sigma_spatial=3.0, sigma_intensity=25.6
# cv2.imwrite('filtered.png', filtered)
```

In [5]:
```
filtered = modified_bilateral(noisy_img, d=9, sigma_spatial=50, sigma_intensity=5
```

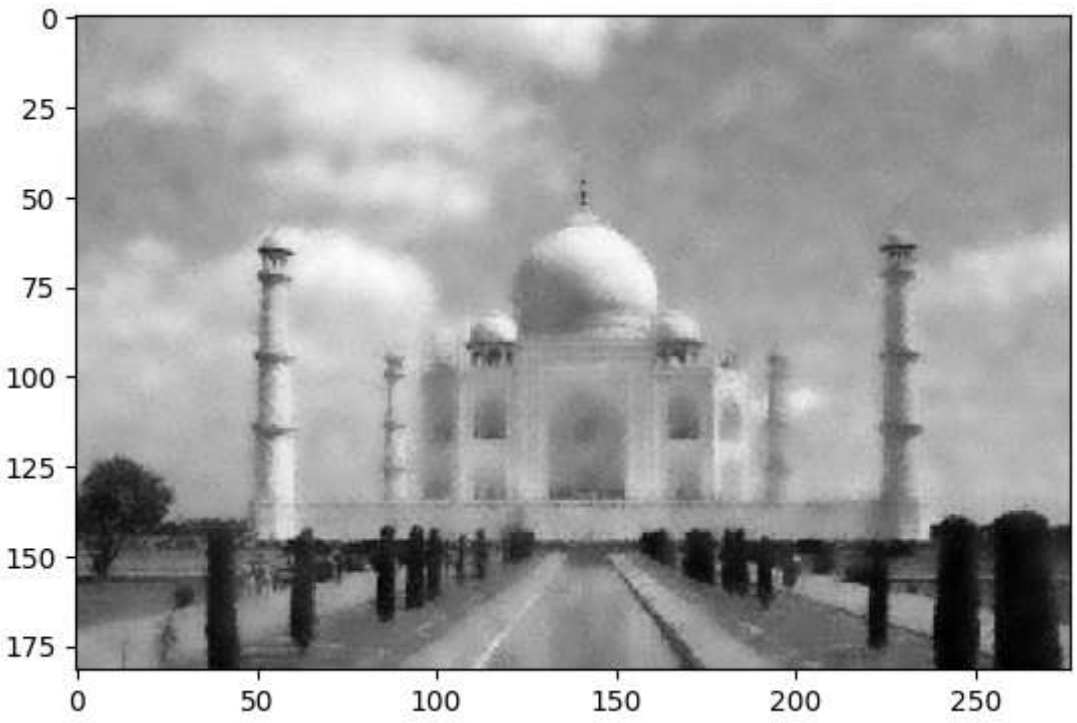
In [6]:
```
plt.imshow(filtered,cmap = "gray")
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x2863785a020>
```



In [7]:
```
plt.imshow(bilateral,cmap = "gray")
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x28639a967a0>
```



In [10]:
```
image_covariance(img, bilateral)/ (img.std() * bilateral.std())
```

Out[10]:
```
np.float64(0.9727615359190009)
```

In [11]:
```
image_covariance(img, filtered)/ (img.std() * filtered.std())
```

Out[11]:
```
np.float64(0.9501200130440134)
```

In [ ]:
```

```