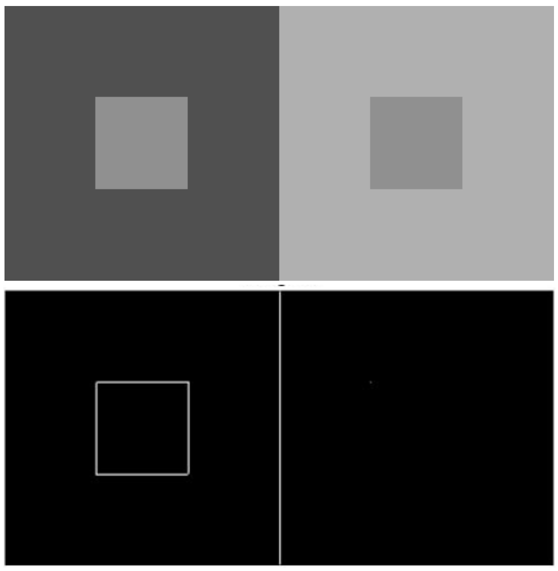
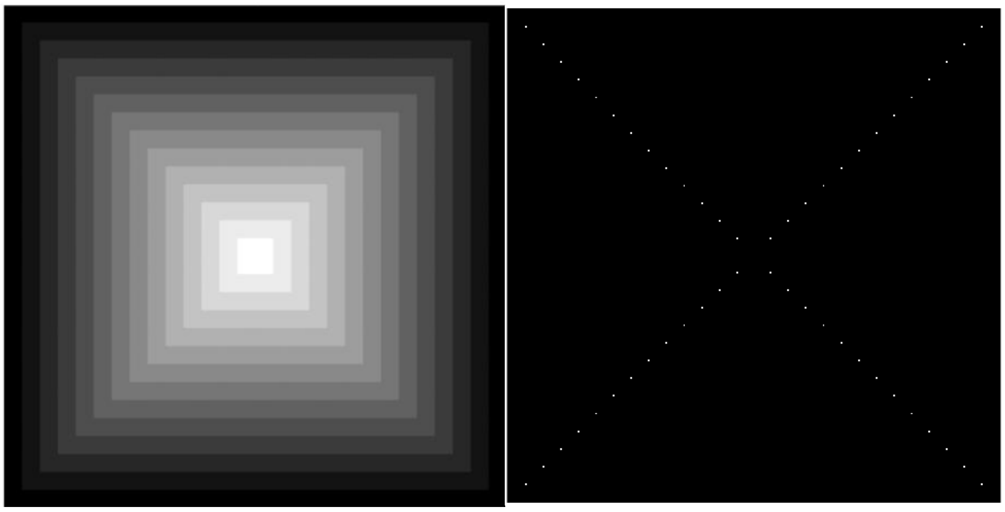
Assignment 3 – The Retinex Algorithm

1. image_derivatives(image):
   ```
   image = image.astype(np.float32)
   ix = convolve2d(image, kx, mode='full')
   iy = convolve2d(image, ky, mode='full')
   ix = ix[:,:-1]
   iy = iy[:-1,:]
   ix[0,:] = 0
   ix[-1,:] = 0
   ix[:,0] = 0
   ix[:,-1] = 0
   iy[0,:] = 0
   iy[-1,:] = 0
   iy[:,0] = 0
   iy[:,-1] = 0
   return ix, iy
   ```
2. def deriv2laplace(ix, iy):
   ```
   ix2 = convolve2d(ix, kx, mode='same')
   iy2 = convolve2d(iy, ky, mode='same')
   return ix2 + iy2
   ```
3. The square on the left seems a little brighter because of the higher contrast with the background. I applied a threshold T to the absolute value of the Laplacian image to identify only the strongest contrast regions. When this threshold is set appropriately, we find that pixels exceeding the threshold appear almost exclusively around the left square.
   the original image is the top image and the image after applying the threshold is the bottom one:

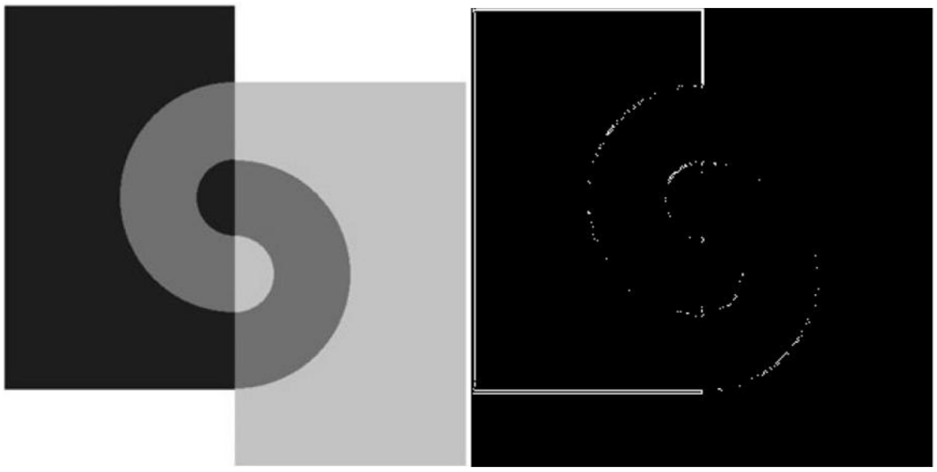


4. The diagonal cross in the image looks brighter than the rest of the shape in each square. The Laplacian image. The image to the left is the original and the image to the right is the Laplacian after applying the threshold.



5. When we compute the Laplacian of the image and apply a threshold TTT to its absolute value, we find that the **edges of the ring are equally strong on both sides**. The thresholded binary Laplacian shows **no asymmetry**: both the left and right borders of the ring are equally detected.

   This result contrasts sharply with the previous two images, where a suitable threshold TTT isolated strong edge responses only in the regions that appeared perceptually lighter. In this case, no such threshold exists that explains the illusion — the Laplacian does not distinguish between the two halves of the ring.



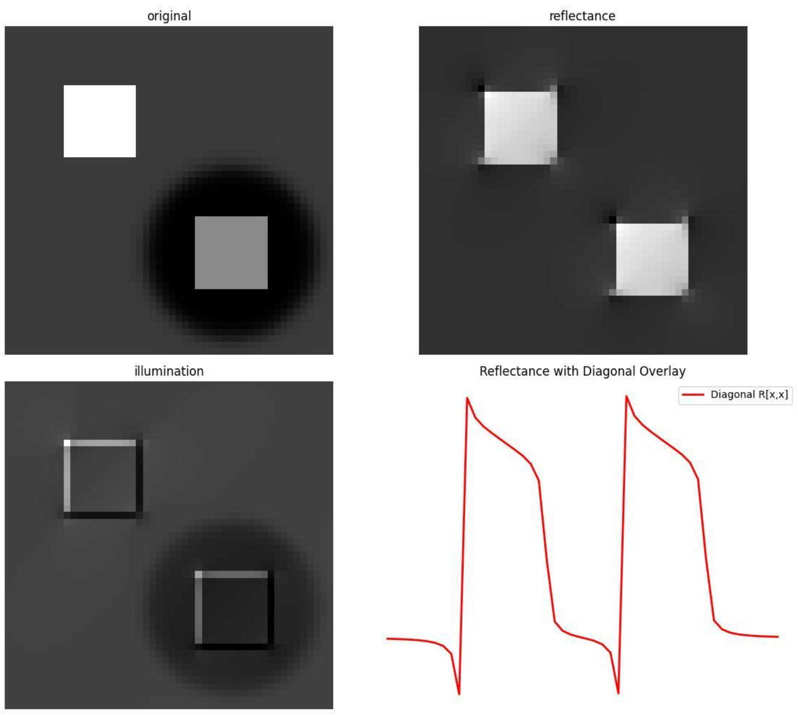
6. In the first two illusions (the squares and the cross), the Laplacian successfully highlights areas with stronger contrast, and a suitable threshold isolates regions that align with where the illusion is perceived. This suggests that early visual processing such as contrast detection via mechanisms similar to the Laplacian, contributes to the perception of color in these cases.

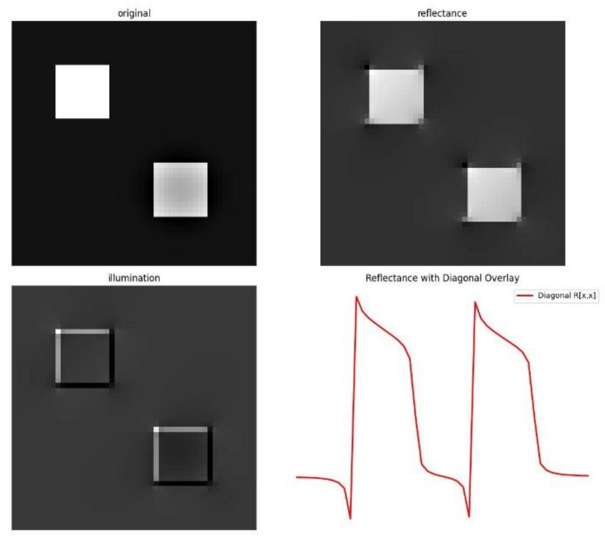
   However, in the Koffka ring, the Laplacian shows no difference between the two halves of the ring. A single threshold does not distinguish the part that appears lighter from the part that appears darker. This indicates that only local contrast cannot explain the illusion. This suggests involvement of additional mechanisms on top of the Laplacian-based one.

An alternative hypothesis can be a form of spatial normalization where perceived color is affected not only by local contrast but also by an overall distribution of lightness across regions in the image.

7.
```python
def do_retinex(image, threshold):
    log_image = get_image_log(image)
    log_ix, log_iy = image_derivatives(log_image)
    log_der_norm = calculate_norm(log_ix, log_iy)
    mask = log_der_norm >= threshold
    filtered_ix = mask * log_ix
    filtered_iy = mask * log_iy
    reflectance_laplacian = deriv2laplace(filtered_ix,
filtered_iy)
    inv_lap_k = inv_del2(image.shape)
    log_reflectance = convolve2d(reflectance_laplacian,
inv_lap_k, mode='same')
    reflectance = np.exp(log_reflectance)
    illumination = image / (reflectance + 1e-8)
    return reflectance, illumination
```

8. When viewing the reflectance output R using, both squares appear nearly identical in brightness. To verify this, we extract the diagonal elements $R[x, x]$ and plot them as a 1D function. The resulting plot shows no significant difference in reflectance values across the two squares.



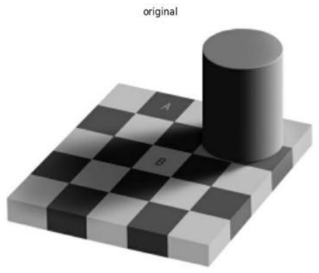


The reflectance image R shows that the bottom square is slightly darker than the top one, especially toward its center. When we plot the diagonal values $R[x,x]$, we observe that the reflectance curve dips in the region corresponding to the blob, though the peak reflectance of the bottom square is nearly the same as that of the top square.

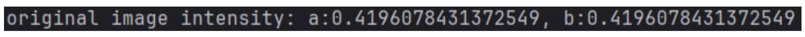
This behavior indicates that the Retinex algorithm partially attributes the blob to reflectance, rather than illumination. Unlike in $twoSquares(1)$, where the smooth gradient occurred in the background, the blob here is within an object. Because the blob introduces a local intensity gradient inside a region of uniform background, it violates one of the Retinex assumptions – that illumination affects the scene uniformly and smoothly, while reflectance changes are sharp and localized.

Changing the threshold might help in suppressing the blob in the reflectance but might be problematic because a higher threshold risks removing real reflectance edges in other parts of the image.

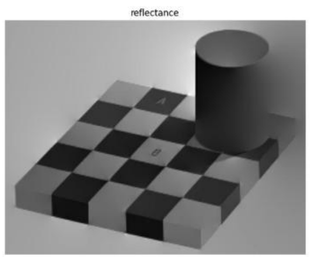
9.



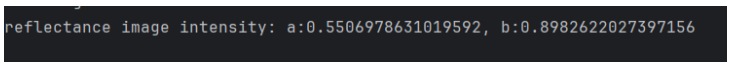
As expected, the two tiles A and B have identical intensity values in the input image. This directly contradicts our visual perception, where tile A appears dark and tile B appears light.



```
original image intensity: a:0.4196078431372549, b:0.4196078431372549
```



In the reflectance image, tile A (in light) appears darker, and tile B (in shadow) appears lighter — aligning with our perceptual interpretation. Thus, the Retinex algorithm correctly "explains" the illusion in terms of reflectance recovery.



```
reflectance image intensity: a:0.5506978631019592, b:0.8982622027397156
```

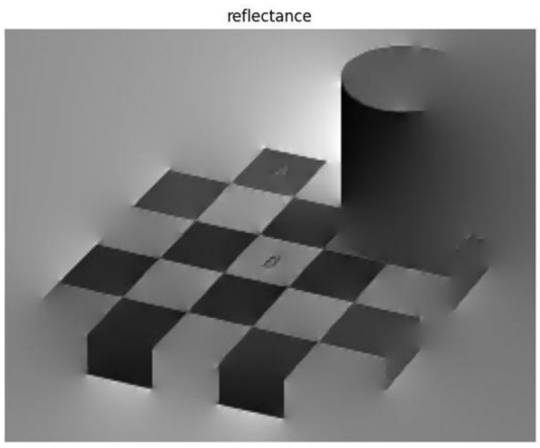
One region where the algorithm fails is the surface of the cylinder casting the shadow. The side facing away from the light appears very dark in the reflectance image — far darker than it should be. This is problematic because the shading change across the cylinder is smooth and gradual, meaning it should have been attributed to illumination, not reflectance. The large contrast leads the algorithm to falsely interpret it as a material change.

This failure occurs because the gradient on the cylinder, though gradual, is large in magnitude — exceeding the threshold and therefore being retained as part of the reflectance.

Would Changing the Threshold Help?
Adjusting the threshold could help in some cases.

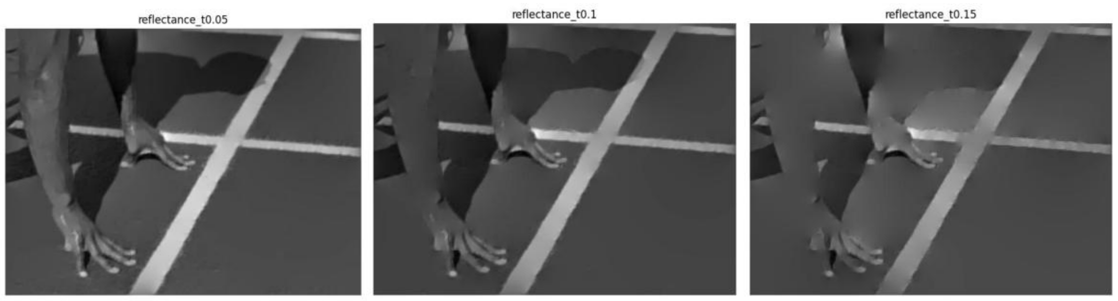
Increasing the threshold might suppress large but smooth gradients like those on the cylinder, causing them to be interpreted as illumination instead. However, doing so would risk removing meaningful reflectance boundaries elsewhere in the image, such as tile edges or object contours.

For example, here is the reflectance image after applying the Retinex algorithm with threshold=0.2:



Thus, while tuning the threshold might improve some regions, it is not a general solution. The limitation lies in the global, fixed threshold and the algorithm's inability to account for smooth but strong illumination gradients. A more robust solution would require adaptive thresholding or incorporating shape and scene priors, which are beyond the basic Retinex model.

10.



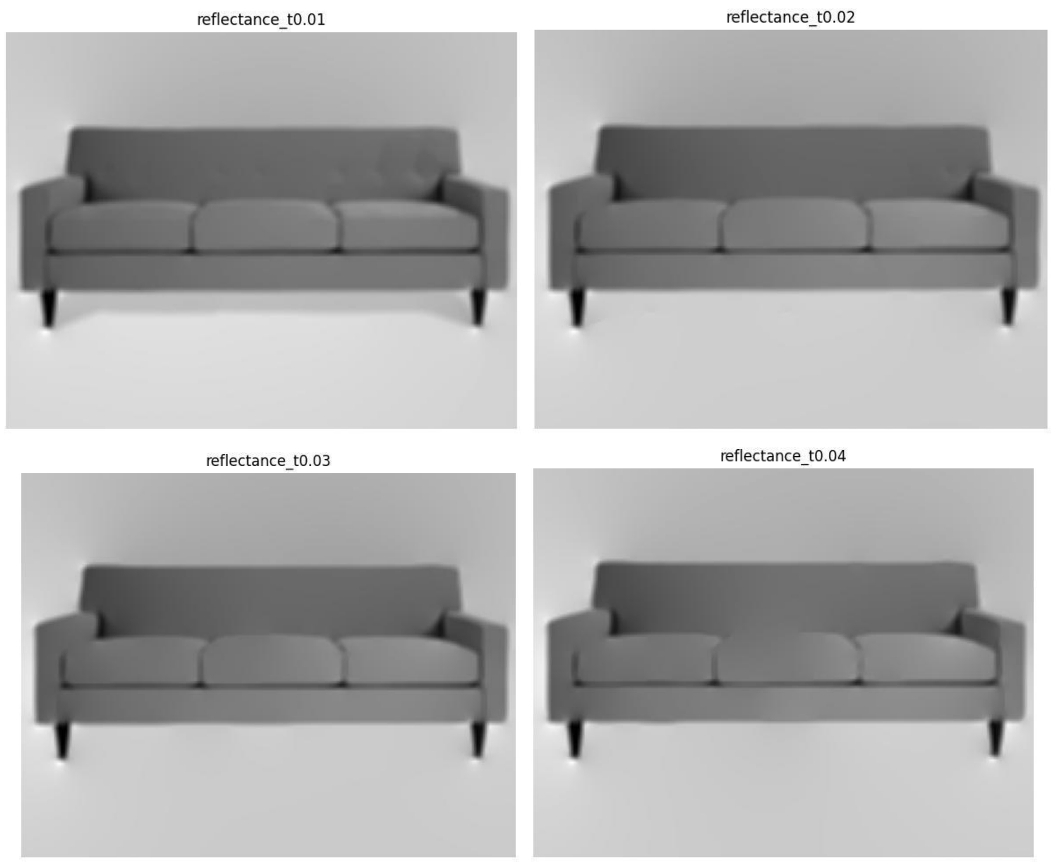
We run the algorithm with thresholds ranging from 0.05 to 0.15, inspecting how well shadows are suppressed:

- At low thresholds ($T = 0.05$), many edges, including shadows, are preserved. The algorithm interprets the sharp shadow boundary as a reflectance edge, failing to remove the shadow.
- As the threshold increases, soft gradients are increasingly suppressed. However, because the shadow edge is very sharp it is still retained in the reflectance image — misclassified as part of the object.
- At high thresholds ($T = 0.15$), the algorithm begins to suppress more edges — but at the cost of removing actual reflectance information. Parts of the runner's skin tone merge with the ground, distorting object boundaries.

Cues that can help an algorithm distinguish the reflectance map from the illumination map:

- Color information – maybe knowing the way colors behave under different illuminations
- Scene geometry of depth map – helps distinguish between different objects
- Texture analysis – detecting texture changes might help distinguish between different objects

11. Results:



- At $T = 0.01$, most small gradients are preserved. The reflectance image still shows mild lighting variations and creases in the couch surface.

- Increasing to $T = 0.02$ and $T = 0.03$ progressively removes more of the illumination details, especially soft shadows and surface texture caused by lighting. The image appears flatter, with more uniform reflectance.

- At $T = 0.04$, even the borders between the pillows begin to disappear. This is arguably acceptable since the pillows are made from the same material, but it may oversimplify the actual structure.

Comparison with the Runner Image
The performance here is qualitatively better than in the previous task (runner.mat) for several reasons:

- The illumination changes in the couch image are smooth and low contrast, fitting the Retinex assumption that illumination varies gradually.
- There are no sharp shadows or cast edges — the light transitions gently across the surface.
- The couch scene consists of relatively flat surfaces with uniform depth and no significant occlusions. In contrast, the runner scene contains more complex geometry with two arms and background at varying depths.
- The couch has a color that is clearly distinct from the light-colored background, making edges easier to distinguish. In the runner image, the skin tone of the runner is similar to the background, which reduces contrast and makes it harder for the algorithm to separate object boundaries from shadow boundaries, especially at higher thresholds.

Code:

```
import cv2
import numpy as np
import matplotlib
# import scipy
import scipy.io as sio
from scipy.fft import fft2, ifft2
from scipy.signal import convolve2d
from scipy.special.cython_special import kl_div
import os

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

kx = np.array([[0.5, -0.5]], dtype=np.float32)
ky = np.array([[-0.5], [0.5]], dtype=np.float32)
plot_save_directory = 'exercise 3/plots'

def show_matlab(im1, sc=None):
    plt.figure()
    if sc is not None:
        plt.imshow(im1, cmap='gray', vmin=sc[0], vmax=sc[1])
    else:
        plt.imshow(im1, cmap='gray')
    plt.axis('image')
    plt.colorbar()
    plt.show()

def show_image(img, title='title'):
    cv2.imshow(title, img)
    cv2.waitKey(0)

def deriv2laplace(ix, iy):
    ix2 = convolve2d(ix, kx, mode='same')
    iy2 = convolve2d(iy, ky, mode='same')
    return ix2 + iy2

def image_derivatives(image):
    image = image.astype(np.float32)
    ix = convolve2d(image, kx, mode='full')
    iy = convolve2d(image, ky, mode='full')
    ix = ix[:,:-1]
    iy = iy[:-1,:]
    ix[0,:] = 0
    ix[-1,:] = 0
    ix[:,0] = 0
    ix[:,-1] = 0
    iy[0,:] = 0
    iy[-1,:] = 0
    iy[:,0] = 0
    iy[:,-1] = 0
    return ix, iy

def load_image_grayscale(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE).astype(np.uint8)
    return image

def create_test_image(size=100, thickness=3):
    image = np.zeros((size, size), dtype=np.uint8)
    cv2.line(image, (0,0), (size-1, size-1), color=255, thickness=thickness)
    return image

def make_binary_image(image, threshold):
    binary = np.where(np.abs(image) > threshold, 255, 0)
    return binary

def show_two_images(img1, img2, title, cmap='gray'):
    fig, axs = plt.subplots(2, 1, figsize=(5, 10))

    axs[0].imshow(img1, cmap=cmap)
    axs[0].axis('off')
    axs[0].set_title('Image 1')

    axs[1].imshow(img2, cmap=cmap)
    axs[1].axis('off')
    axs[1].set_title('Image 2')
    plot_path = os.path.join(plot_save_directory, f'{title}.jpg')
    plt.tight_layout()
    plt.savefig(plot_path)
    print(f'saved to {plot_path}')
    plt.show()

def get_image_log(image):
    image = image.astype(np.float32)
    log_image = np.log(image + 1e-6)
    return log_image

def calculate_norm(i1, i2):
    magnitude = np.sqrt(i1 ** 2 + i2 ** 2)
    return magnitude


def soft_thresh(x, t, s=5):
    y = 1 / (1 + np.exp(-s * (x - t)))
    return 1 - y
```

```
def two_squares(shadow_flag):

    R = np.ones((50, 50))
    R[29:40, 29:40] = 2  # MATLAB is 1-indexed; Python is 0-indexed
    R[9:20, 9:20] = 2

    x, y = np.meshgrid(np.arange(1, 51), np.arange(1, 51))
    rr = (x - 35) ** 2 + (y - 35) ** 2

    if shadow_flag == 1:
        L = 1 - 0.3 * soft_thresh(rr, 13**2, 0.05)
    else:
        L = 1 - 0.3 * soft_thresh(rr, 3**2, 0.04)

    im = R * L
    return im

def inv_del2(im_size):
    isize = 2 * max(im_size)
    K = np.zeros((isize, isize), dtype=np.float32)
    center = isize // 2
    K[center, center] = -4
    K[center + 1, center] = 1
    K[center - 1, center] = 1
    K[center, center + 1] = 1
    K[center, center - 1] = 1

    Khat = fft2(K / 4.0)

    Khat_safe = Khat.copy()
    Khat_safe[np.abs(Khat) < 1e-10] = 1
    invKhat = 1.0 / Khat_safe
    invKhat[np.abs(Khat) < 1e-10] = 0

    invK = np.real(ifft2(invKhat))

    shift_kernel = np.array([[1, 0, 0],
                             [0, 0, 0],
                             [0, 0, 0]], dtype=np.float32)
    invK = convolve2d(invK, shift_kernel, mode='same')
    return invK

def do_retinex(image, threshold):
    log_image = get_image_log(image)
    log_ix, log_iy = image_derivatives(log_image)
    log_der_norm = calculate_norm(log_ix, log_iy)
    mask = log_der_norm >= threshold
    filtered_ix = mask * log_ix
    filtered_iy = mask * log_iy
    reflectance_laplacian = deriv2laplace(filtered_ix, filtered_iy)
    inv_lap_k = inv_del2(image.shape)
    log_reflectance = convolve2d(reflectance_laplacian, inv_lap_k, mode='same')
    reflectance = np.exp(log_reflectance)
    illumination = image / (reflectance + 1e-8)
    return reflectance, illumination

def do_retinex_multiple_thresholds(image, thresholds):
    print(f'starting retinex')
    kl_by_reflectance = {}
    if len(thresholds) > 1:
        for t in thresholds:
            reflectance, illumination = do_retinex(image, t)
            kl_by_reflectance[t] = (reflectance, illumination)
        return kl_by_reflectance
    if len(thresholds) == 1:
        reflectance, illumination = do_retinex(image, thresholds[0])
        return reflectance, illumination


def plot_diagonal(image, title='Diagonal Intensity Profile'):

    diag = np.diagonal(image)

    plt.figure()
    plt.plot(diag, marker='o')
    plt.title(title)
    plt.xlabel('Pixel index along diagonal (x)')
    plt.ylabel('Intensity R[x, x]')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def show_3_images_and_diagonal_overlay(images, titles, reflectance, title, cmap='gray'):
    if len(images) != 3 or len(titles) != 3:
        raise ValueError("Expected exactly 3 images and 3 titles.")
    plot_path = os.path.join(plot_save_directory, f'{title}.jpeg')
    plt.figure(figsize=(12, 10))

    for i in range(3):
        plt.subplot(2, 2, i + 1)
        plt.imshow(images[i], cmap=cmap)
        plt.title(titles[i])
        plt.axis('off')

    plt.subplot(2, 2, 4)
    diag = np.diagonal(reflectance)
    plt.plot(np.arange(len(diag)), diag, color='red', linewidth=2, label='Diagonal R[x,x]')
    plt.title('Reflectance with Diagonal Overlay')
    plt.grid(True)
    plt.axis('off')
    plt.legend()

    plt.tight_layout()
    plt.savefig(plot_path)
    plt.show()

def show_image_with_dots(image, x1, y1, x2, y2, dot_radius=5,
                         cmap='gray', title='Image with Red Dots'):
    plt.figure(figsize=(6, 6))
    plt.imshow(image, cmap=cmap)
    plt.scatter([x1, x2], [y1, y2], s=dot_radius**2,
                edgecolors='red', facecolors='none', linewidths=2)
    plt.title(title)
    plt.axis('off')
    plt.show()
    print('finished')

def save_plot(image, filename, title, cmap='gray', vmin=None, vmax=None):
    plt.figure(figsize=(6, 6))
    if image.ndim == 2:
        plt.imshow(image, cmap=cmap, vmin=vmin, vmax=vmax)
    else:
        plt.imshow(image)
    file_path = os.path.join(plot_save_directory, f'{filename}.jpeg')
    plt.title(title)
    plt.axis('off')
    plt.tight_layout()
    plt.savefig(file_path, bbox_inches='tight', pad_inches=0.1)
    plt.close()

def q3():
    path_two_squares = 'exercise 3/ex3-files/simul_cont_squares.tif'
    grayscale_image = load_image_grayscale(path_two_squares)
    ix, iy = image_derivatives(grayscale_image)
    laplacian = deriv2laplace(ix, iy)
    threshold = 15
    binary = make_binary_image(laplacian, threshold=threshold)
    show_two_images(grayscale_image, binary, f'simultaneous_contrast_t{str(threshold)}')

def q4():
    path_cross = 'exercise 3/ex3-files/cross.tif'
    grayscale_image = load_image_grayscale(path_cross)
    ix, iy = image_derivatives(grayscale_image)
    laplacian = deriv2laplace(ix, iy)
    threshold = 4.5
    binary = make_binary_image(laplacian, threshold=threshold)
    show_two_images(grayscale_image, binary, f'cross_t{str(threshold)}')

def q5():
    path_kofkaring = 'exercise 3/ex3-files/kofka_ring.tif'
    grayscale_image = load_image_grayscale(path_kofkaring)
    ix, iy = image_derivatives(grayscale_image)
    laplacian = deriv2laplace(ix, iy)
    threshold = 17
    binary = make_binary_image(laplacian, threshold=threshold)
    show_two_images(grayscale_image, binary, f'kofka_ring_t{str(threshold)}')

def q8():
    titles = ['original', 'reflectance', 'illumination']
    flag = 2
    grayscale_image = two_squares(flag)
    threshold = 0.3
    reflectance, illumination = do_retinex_multiple_thresholds(grayscale_image, [threshold])
    show_3_images_and_diagonal_overlay([grayscale_image, reflectance, illumination],
                                       titles, reflectance, title=f'squares{flag}_t{str(threshold)}', cmap='gray')

def q9():
    mat = sio.loadmat('exercise 3/ex3-files/checkerShadow.mat')
    im1 = mat['im1']
    print(im1.shape)
    coordinates = mat['x1'], mat['y1'], mat['x2'], mat['y2']
    coordinates = [c.item() for c in coordinates]
    x1, y1, x2, y2 = coordinates
    threshold = 0.2
    print(f'original image intensity: a:{im1[y1, x1]}, b:{im1[y2, x2]}')
    reflectance, illumination = do_retinex_multiple_thresholds(im1, [threshold])
    save_plot(reflectance, f'checkerShadow_t{str(threshold)}', f'reflectance_t{str(threshold)}')
    print(f'reflectance image intensity: a:{reflectance[y1, x1]}, b:{reflectance[y2, x2]}')

def q10():
    mat = sio.loadmat('exercise 3/ex3-files/runner.mat')
    im1 = mat['im1']
    print(im1.shape)
    thresholds = [0.05, 0.1, 0.15]
    kl_by_reflectance = do_retinex_multiple_thresholds(im1, thresholds)
    for t in thresholds:
        reflectance, illumination = kl_by_reflectance[t]
        save_plot(reflectance, f'runner_t{str(t)}', f'reflectance_t{str(t)}')

def q11():
    mat = sio.loadmat('exercise 3/ex3-files/couch.mat')
    im1 = mat['im1']
    thresholds = [0.01, 0.02, 0.03, 0.04]
    kl_by_reflectance = do_retinex_multiple_thresholds(im1, thresholds)
    for t in thresholds:
        reflectance, illumination = kl_by_reflectance[t]
        save_plot(reflectance, f'couch_t{str(t)}', f'reflectance_t{str(t)}')

def main():
    q11()

if __name__ == "__main__":
    main()
```