Wavelets Assignment

Serkan Shentyurk
Student ID: #r0833419

Section 2 - Wavelet-Based Denoising

2.1 - A Univariate Functions with Noise

Question 2.1

The function $f(x)$ is given, and 200 equally spaced data points have been generated between -2 and 2

$$f(x) = (2 + \cos(x)) \cdot |x| \cdot \text{sign}(x - 1) \tag{1}$$

The $(x, f(x))$ pairs undergo a wavelet transformation using a 2-level DB2 wavelet with symmetric padding at the boundaries. The function and wavelet coefficients are depicted in Figure 1.

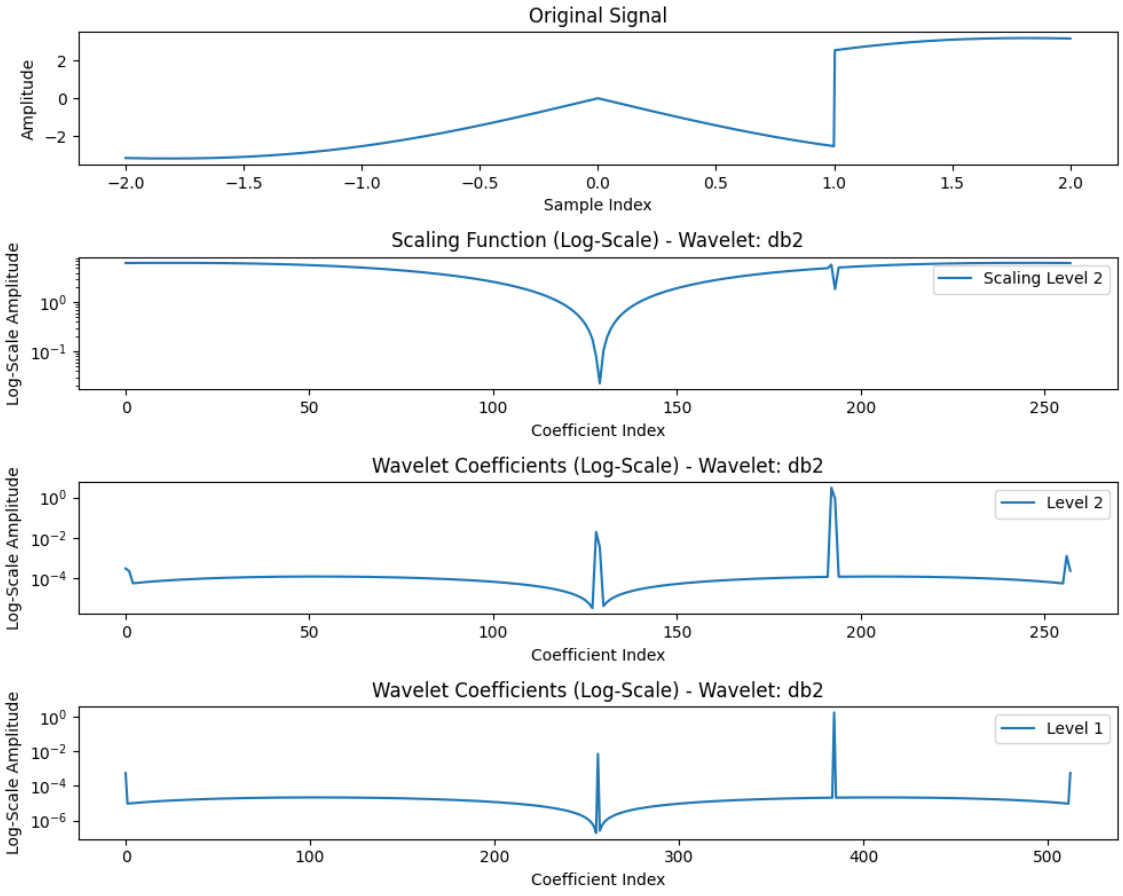


Figure 1: The Wavelet Transformation of $f(x)$ with DB2 wavelet - Level 2.

Large coefficients may indicate significant changes or details in the corresponding region of the signal. There is drastic changes within the signal at $x = -2$, $x = 0$, $x = 1$, and $x = 2$ (Figure 1). We can see peaks at the wavelet coefficients at Levels 1 and 2: We have peaks at -2, 0, 1, and 2. These peaks indicated significant changes in the region, which can be seen in Figure 1. Since other regions of the signal are smooth, the coefficients are relatively small. As the level increases, it is seen that that drastic change is captured better than the lower levels.

Question 2.2

Noise, denoted as ϵ, is introduced to the signal that follows a normal distribution with a mean of 2 and a variance of 1, affecting the function $f(x)$ to yield the modified function $\tilde{f}$.

$$\tilde{f} = f + \epsilon, \quad \epsilon \sim \mathcal{N}(\mu = 2, \sigma^2 = 1) \tag{2}$$

The wavelet transform of $\tilde{f}$ was computed using DB2 with 8 levels and symmetric padding. Noise in the wavelet domain was subsequently removed through the application of soft and hard thresholding with a threshold of 1. The signal was then transformed back, as illustrated in Figure 2.

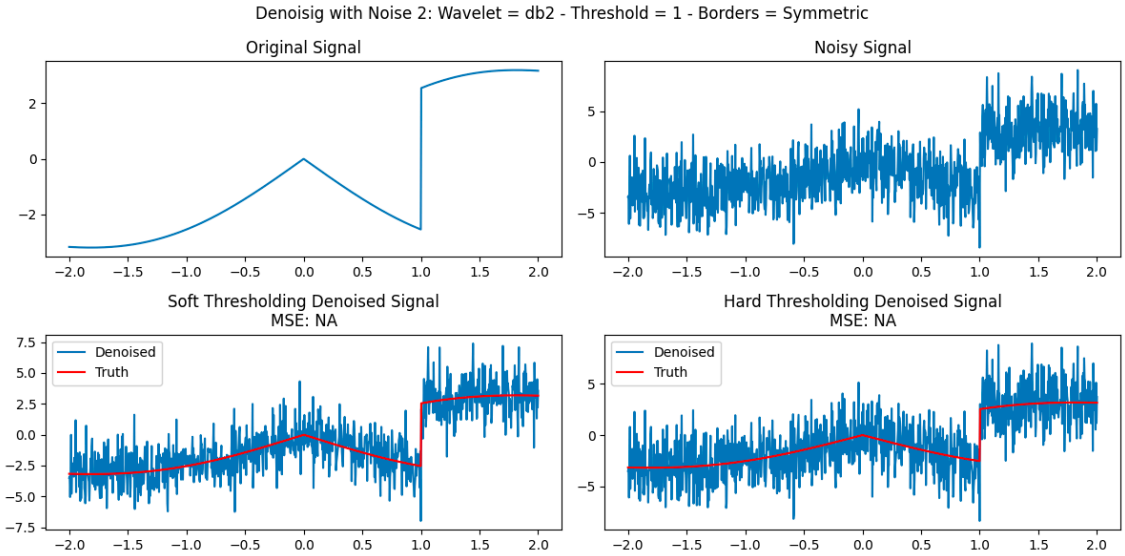


Figure 2: Denoising of $\tilde{f}$ with Wavelet = DB2, Threshold = 1 and Symmetric Padding.

The denoising scheme is not optimized, resulting in a lack of smooth performance. However, under these settings, soft thresholding appears to outperform hard thresholding, effectively reducing fluctuations.

Question 2.3

The results can be compared through an examination of the mean squared error (MSE) of the transformed signal.

$$MSE(\tilde{f}(x), f(x)) = \Sigma_i(\tilde{f}(x_i) - f(x_i)^2) \tag{3}$$

The objective was to minimize the mean squared error (MSE) by experimenting with various thresholds δ and different levels l for the wavelet transform using DB2.

$$L = [1, 2, 3, ..., 8] \tag{4}$$

$$\Delta = [0.5, 1.0, 1.5, ..., 10.0] \tag{5}$$

$$l, \delta = argmin_{l,\delta}\, MSE(\tilde{f}(x), f(x)), \qquad l \in L, \delta \in \Delta \tag{6}$$

The minimum Mean Squared Error (MSE) of 0.186 (Figure 3) is achieved by employing the wavelet db2 in conjunction with hard thresholding. This optimal result is attained with a threshold setting of 7.5, utilising a level-8 decomposition and symmetric padding.

The boundaries and edges of a signal are challenging for denoising algorithms since they contain sudden changes and sharp transitions. They are more prone to artefacts. When the denoised signal is checked in Figure 3, it is seen that the algorithm works relatively well and only deviates at $x = 1$. However, it is important to note that this denoising scheme does not output the function and/or distribution of the original signal, hence it is not possible to get the exact original signal back.

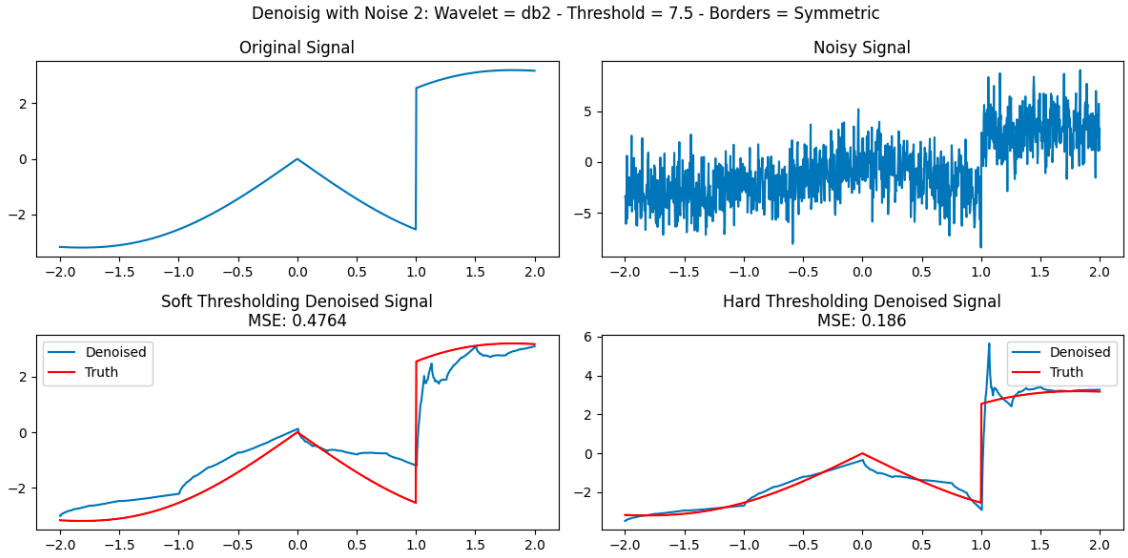


Figure 3: Optimised Denoising Scheme of $\tilde{(f)}$. Wavelet = DB2, Threshold = 7.5, Level = 8, Symmetric Padding.

2.2 - Images with Noise

Question 2.4

In this section, an image (Figure 4a) serves as the dataset. We introduced noise, denoted as ϵ, which adheres to a normal distribution with a mean of 20 and a variance of 1, to the image f, resulting in the creation of the noisy image $\tilde{f}$ (Figure 4b).

$$\tilde{f} = f + \epsilon, \quad \epsilon \sim \mathcal{N}(\mu = 20, \sigma^2 = 1) \tag{7}$$

(a) Original Image (b) Noisy Image

Figure 4: Sample Image 1.

Question 2.5

To evaluate the efficacy of denoising, signal-to-noise ratio (SNR) is employed as an alternative to mean squared error. Daubechies 2, Symlets 4, and Coiflets 2 wavelets were utilized (Figure 12), employing both soft and hard thresholding techniques (Figure 5), and SNR values were computed (Table 1).

Table 1: SNR Values of Different Denoising Schemes

Wavelet	Threshold	Threshold Number	Padding	Level	Redundant	SNR
Daubechies 2	Hard Thresholding	20	Symmetric	3	False	17.2005
Daubechies 2	Soft Thresholding	20	Symmetric	3	False	22.0548
Symlets 4	Hard Thresholding	20	Symmetric	3	False	17.2269
Symlets 4	Soft Thresholding	20	Symmetric	3	False	22.2197
Coiflets 2	Hard Thresholding	20	Symmetric	3	False	17.2333
Coiflets 2	Soft Thresholding	20	Symmetric	3	False	22.2797
Daubechies 2	Hard Thresholding	20	Symmetric	3	True	16.5220
Daubechies 2	Soft Thresholding	20	Symmetric	3	True	16.5455
Symlets 4	Hard Thresholding	20	Symmetric	3	True	16.5076
Symlets 4	Soft Thresholding	20	Symmetric	3	True	16.525
Coiflets 2	Hard Thresholding	20	Symmetric	3	True	16.5062
Coiflets 2	Soft Thresholding	20	Symmetric	3	True	16.5233

SNR values exceeding 20 are regarded as favourable approximations. Upon comparing SNR values, it becomes evident that soft thresholding outperforms hard thresholding, yielding higher SNR values and better images (Figure 5). Specifically, hard thresholding exhibits SNR values surpassing 20, while soft thresholding displays SNR values below 20 (Table 1). However, no significant variance is observed in the outcomes when employing different wavelets (Table 1, Figure 5).

(a) Daubechies 2 Hard Thresholding

(b) Daubechies 2 Soft Thresholding

(c) Symlets 4 Hard Thresholding

(d) Symlets 4 Soft Thresholding

(e) Coiflets 2 Hard Thresholding

(f) Coiflets 2 Soft Thresholding

Figure 5: Denoising with Different Thresholding and Wavelets

2.3 - Using a Redundant Wavelet Transform

Question 2.6

The SWT function from the PyWavelets library is employed to implement a wavelet-based denoising scheme utilising the redundant wavelet transform.

Question 2.7

Denoising using the redundant wavelet transformation is depicted in Figure 6, and corresponding SNR values are calculated and presented in Table 1.

When employing redundant wavelet transformation, minimal variations are observed across different wavelets and thresholding methods. In comparison to the standard wavelet transform, the results indicate consistent SNR values for hard thresholding, while a notable decrease is observed in the SNR values of the soft thresholding scheme (Table 1). Interestingly, the standard wavelet transform with soft thresholding outperforms other schemes in terms of denoising effectiveness.

Question 2.8

The same problem in Question 2.1 is repeated with redundant wavelet transformation with symmetric padding, again optimising for threshold and level (Figure 7). The minimum Mean Squared Error (MSE) of 1.7036 (Figure 7) is achieved by employing the wavelet db2 in conjunction with soft thresholding. This optimal result is attained with a threshold setting of 4.5, utilising a level-3 decomposition and symmetric padding.

The MSE of the redundant wavelet transform 1.7036, is drastically higher than the standard wavelet transform's MSE, 0.186. Hence, redundant wavelet transformation performed worse in the current setting.

Question 2.9

The grid lines introduced every tenth row and every tenth column in the image, have been attempted to be removed through a denoising scheme. The denoising process involved both Discrete Wavelet Transformation and Redundant Wavelet Transformation, each conducted with a level of 5 and a threshold of 50. The outcome of this denoising effort is illustrated in Figure 8.

When applying redundant wavelet transformation, it is evident that, in both soft and hard thresholding scenarios, the method falls short in effectively removing the grid lines. Furthermore, the resulting images exhibit blurriness to the extent that direct interpretation becomes challenging (Figure 8). Comparatively, standard wavelet transformation performs better than the redundant approach; however, residual grid lines persist. Although there are improvements with hard thresholding, the lines remain visibly present.

Due to the suboptimal reconstruction quality intentionally pursued (as indicated by the undesirable visual artefacts), SNR values are not computed or provided in this context.

Section 3 - Wavelet-Based Inpainting

Question 3.1

Denoising schemes were applied with varying soft and hard thresholding, utilizing threshold values denoted as Δ, employing different wavelets denoted as W, and considering different levels denoted as L,

(a) Daubechies 2 Hard Thresholding

(b) Daubechies 2 Soft Thresholding

(c) Symlets 4 Hard Thresholding

(d) Symlets 4 Soft Thresholding

(e) Coiflets 2 Hard Thresholding

(f) Coiflets 2 Soft Thresholding

Figure 6: Denoising with Different Redundant Wavelet Transformation Schemes

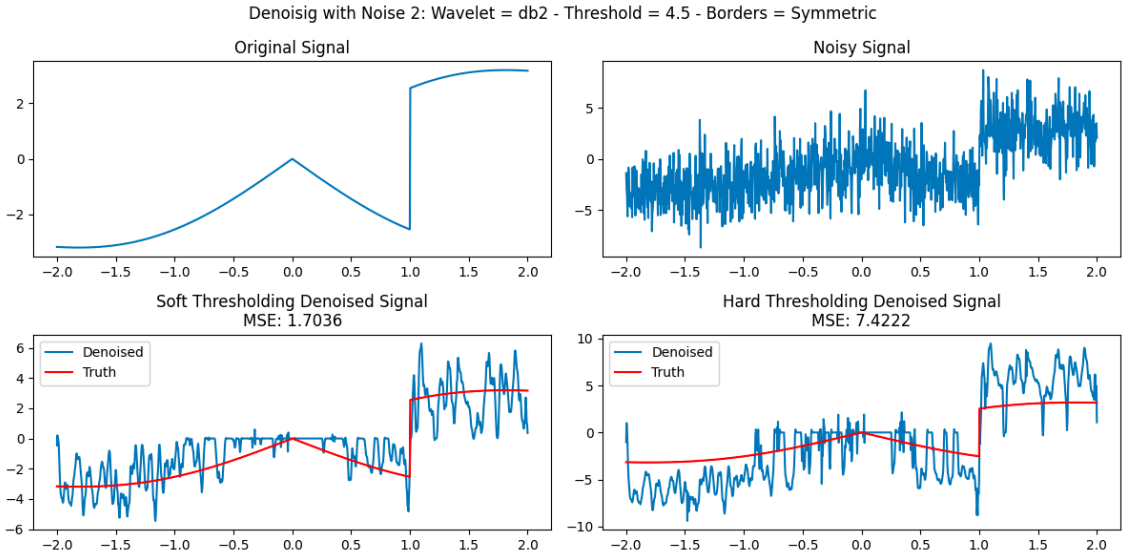


Figure 7: Optimised Denoising Scheme of $\tilde{f}$. Wavelet = DB2, Threshold = 4.5, Level = 3, Symmetric Padding

each subjected to 1000 iterations.

$$L = [1, 2, 3, 4, 5] \tag{8}$$

$$\Delta = [10, 20, ..., 100] \tag{9}$$

$$W = [db2, sym4, coif2] \tag{10}$$

Such schemes were applied to two different images (Figure 9) and two distinct noise combinations: grid-line noise (Figure 9c) and random pixel noise (Figure 9f). For random pixel noise, an equivalent number of pixels as the grid-line noise model was randomly selected, and their pixel values were set to 0.

The SNR values corresponding to various denoising schemes applied to the sample images are illustrated in Figure 10. In this figure, the blue lines denote the standard wavelet transform, while the red lines signify the redundant wavelet transform. Solid lines indicate soft thresholding, whereas dashed lines represent hard thresholding.

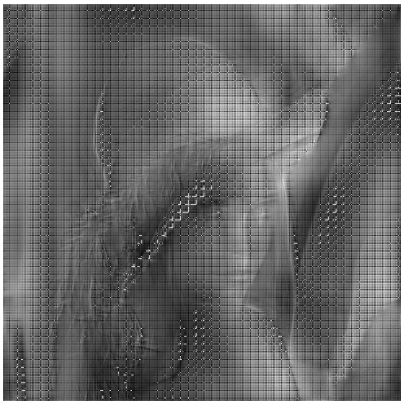
Each subfigure in the plot displays the results of the denoising algorithm for one of the four image-noise combinations. The x-axis of each subfigure is segmented into three areas by two thick black lines, namely DB2, SYM4, and COIF2 (10), each dedicated to a specific wavelet.

Furthermore, each of these subareas is subdivided into ten different segments by nine thin black lines. Each of these 'sub-subareas' corresponds to a distinct threshold level (9). Within each sub-subarea, there are five data points, representing the SNR values for five different levels of wavelet transformation (8).

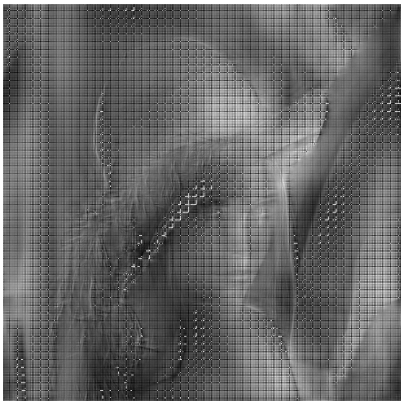
For the first image with random noise, SWT with hard-thresholding the coif2 wavelet, level 1, and 30 thresholds are found to be the best combination that maximises the SNR (Figure 10a). The SNR of the best hyperparameter combination is 35.0878.
For the first image with line noise, SWT with hard-thresholding the sym4 wavelet, level 1, and 30 thresholds are found to be the best combination that maximises the SNR (Figure 10b). The SNR of the best hyperparameter combination is 32.3176.

(a) Sample Image

(b) Line-added Image

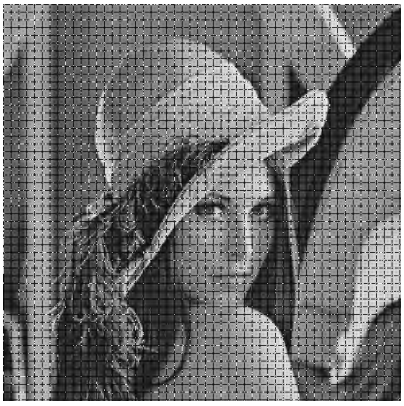


(c) Denoising Scheme with Redundant Wavelet Transformation, Soft Thresholding



(d) Denoising Scheme with Redundant Wavelet Transformation, Hard Thresholding

(e) Denoising Scheme with Discrete Wavelet Transformation, Soft Thresholding



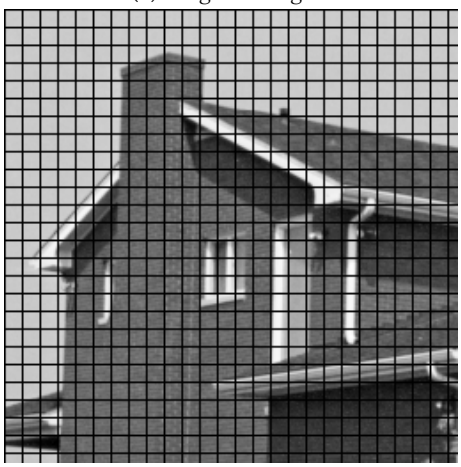
(f) Denoising Scheme with Discrete Wavelet Transformation, Hard Thresholding

Figure 8: Denoising with Different Redundant Wavelet Transformation Schemes. Level:5, Threshold:50

(a) Original Image 1

(b) Original Image 2

(c) Noisy Image 1 - Line

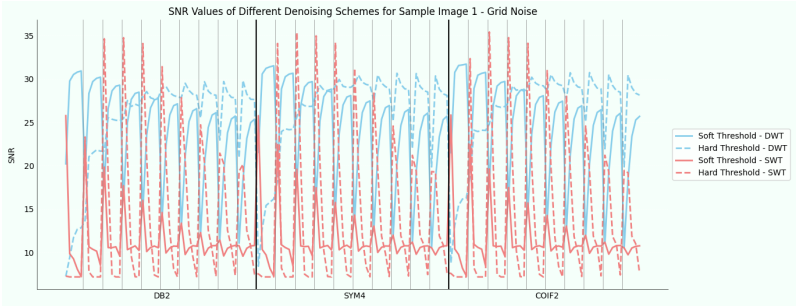


(d) Noisy Image 2 - Line

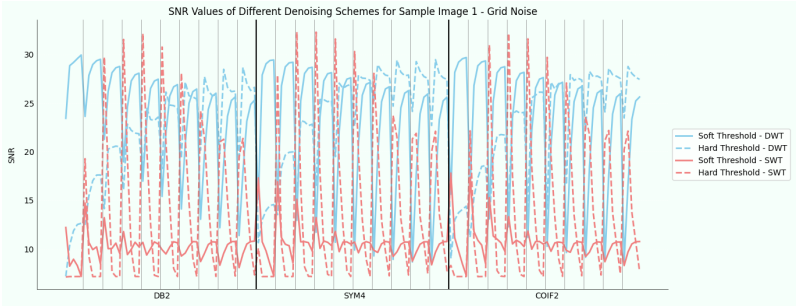
(e) Noisy Image 1 - Random

(f) Noisy Image 2 - Random

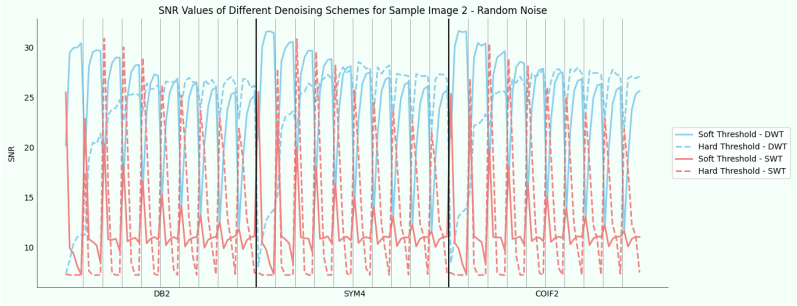
Figure 9: Noisy Images



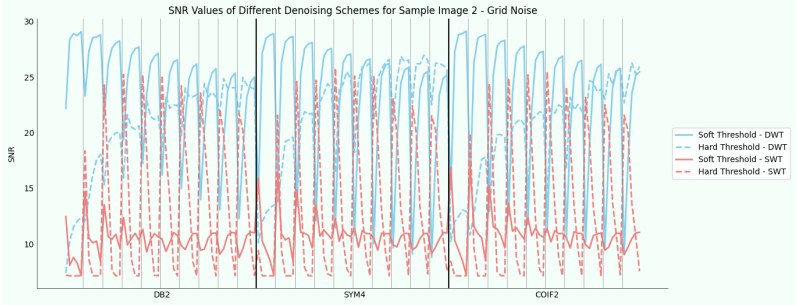
(a) SNR values of different denoising parameters - Image 1 with random noise.



(b) SNR values of different denoising parameters - Image 1 with lines.



(c) SNR values of different denoising parameters - Image 2 with random noise.



(d) SNR values of different denoising parameters - Image 2 with lines.

Figure 10: SNR results of different Wavelet-Level-Threshold combinations.

For the second image with random noise, DWT with soft-thresholding the coif2 wavelet, level 1, and 30 thresholds are found to be the best combination that maximises the SNR (Figure 10c). The SNR of the best hyperparameter combination is 30.1020.
For the second image with line noise, DWT with soft-thresholding the coif2 wavelet, level 5, and 10 thresholds are found to be the best combination that maximises the SNR (Figure 10d). The SNR of the best hyperparameter combination is 29.1400.

These findings are listed in the Table 2.

Image	Wavelet	Threshold	Threshold Level	Level	SNR
Image 1	Coif2	Hard	30	1	35.0878
Image 1	Sym4	Hard	30	1	32.3176
Image 2	Coif2	Soft	30	1	30.1020
Image 2	Coif2	Soft	10	5	29.1400

Table 2: The Overview Results of Tuned Denoising Schemes

The denoising scheme for all 4 images is done with the best hyperparameters. The original image, the noisy image, and the reconstructed image for 4 cases are shown in Figure 11.

Question 3.2

The analysis reveals interesting distinctions in the behaviour of soft and hard thresholding across various denoising scenarios, particularly when considering different wavelet transformations, threshold levels, and noise characteristics.

When comparing the effects of different wavelet transformations (standard or redundant), both soft and hard thresholding exhibit similar behaviour. However, noticeable differences emerge when examining the impact of varying threshold levels. Generally, hard thresholding tends to outperform soft thresholding as the threshold level increases, except for the specific case of Image 2 with Grid Line noise. In this instance, soft thresholding either performs better or on par with hard thresholding.

Moreover, the influence of the level of wavelet transformation introduces another layer of complexity to the comparison. As the level increases, soft thresholding demonstrates improved performance, while hard thresholding tends to exhibit a decline in effectiveness.

The observed exception in Image 2 with Grid Line noise suggests that the effectiveness of thresholding methods can be context-dependent and may vary based on the nature of the noise present in the image.

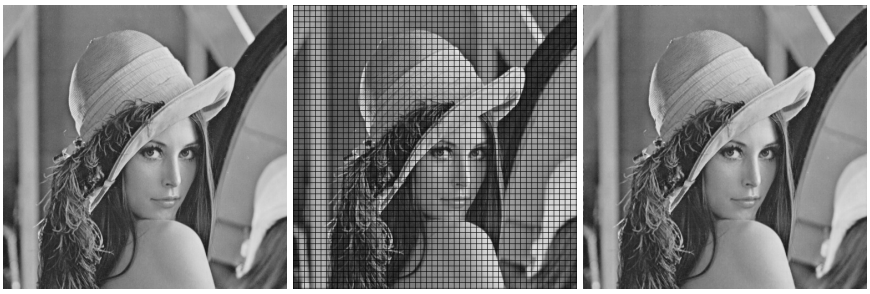
Question 3.3

The observed behaviour in the denoising results indicates distinct patterns between redundant wavelet transformation and standard wavelet transformation, particularly in response to varying threshold levels and thresholding types. However, they behave similarly when different wavelets are used.

For redundant wavelet transformation, there is a slight increase and then a decline in performance as the threshold increases, and beyond a certain point, the SNR values tend to stabilise. This trend holds for both soft and hard thresholding techniques (Figure 10).

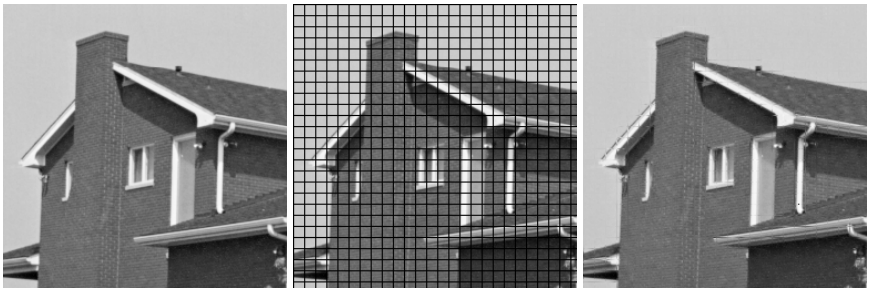
On the other hand, the standard wavelet transformation exhibits differential responses based on the thresholding method employed. In the case of soft thresholding, there is a slight decrease in SNR values as

(a) Original Image (b) Noisy Image (c) Denoised Image



(d) Original Image (e) Noisy Image (f) Denoised Image

(g) Original Image (h) Noisy Image (i) Denoised Image



(j) Original Image (k) Noisy Image (l) Denoised Image

Figure 11: Origian Image - Noisy Image - Denoised Image.

the threshold level increases. Conversely, with hard thresholding, there is an increase in SNR values as the threshold level rises, and then there is a decrease after a certain point (Figure 10).

Additionally, the impact of increasing the level of wavelet transformation varies between redundant and standard wavelet transformations. Redundant wavelet transformation tends to perform better at lower levels, and its performance decreases as the level increases. In contrast, there are no substantial changes in the performance of standard wavelet transformation across different levels (Figure 10).

It is seen that such behavioural trends persist across given example images. However, considering the differences, even if they are small, it is better not to generalise to other denoising problems.

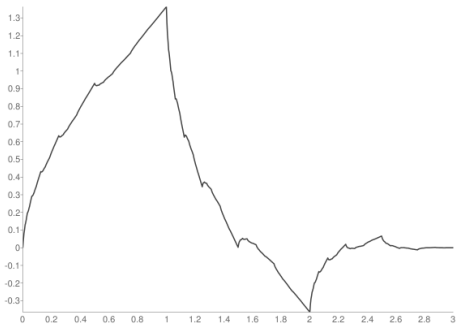
Question 3.4

When examining various wavelets across all denoising scenarios, it becomes evident that different wavelets yield relatively similar SNR values when subjected to the same threshold level, thresholding type (soft/hard), and level (Figure 10). This phenomenon exists across examples. This observed similarity in SNR values could potentially be attributed to the convergence of wavelets within 1000 iterations.

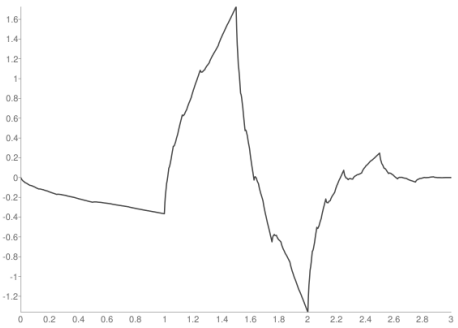
It is worth noting that there might be variations in the number of iterations required to achieve optimal SNR values for different wavelets. However, due to the considerable time investment — approximately 8 hours for a single grid-search-like algorithm, and considering there are four such algorithms — the step involving a detailed exploration of iteration counts is intentionally omitted, given the time constraints of less than 2 days.

Appendix

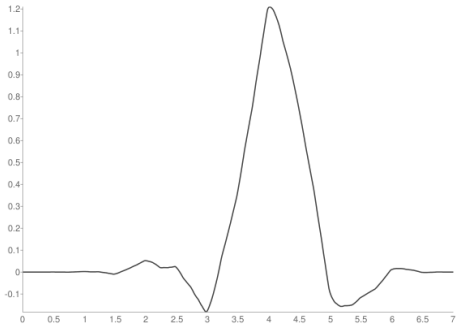
Wavelets and Scaling Functions



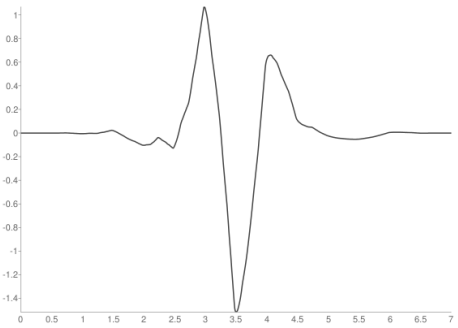
(a) Daubechies 2 Scaling Function



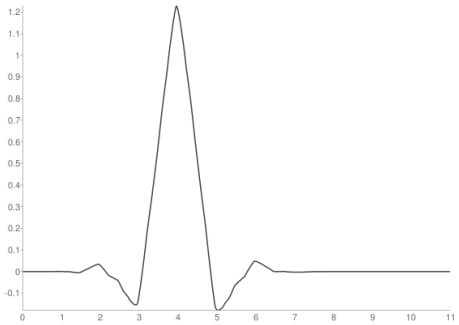
(b) Daubechies 2 Wavelet Function



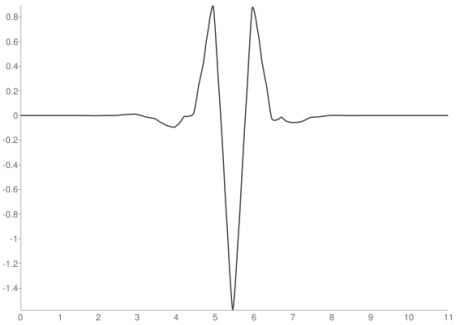
(c) Symlets 4 Scaling Function



(d) Symlets 4 Wavelet Function



(e) Coiflets 2 Scaling Function



(f) Coiflets 2 Wavelet Function

Figure 12: Wavelets and Scaling Functions

Code

There is a Github Repo containing the code to create this work. There can be minor changes (all done before the deadline), the up-to-date version is available on GitHub. The repo will be publicly after the deadline for the project (20th of December).

Util Functions

Some functions are defined to use in the analysis.

```
import matplotlib.pyplot as plt
import numpy as np
import pywt
from PIL import Image

def load_image(path):
    '''
    Load images from the given path.
    Makes sure the image contains even-numbered pixels (for practical reasons)
    Returns the image as numpy array
    '''
    sample_image_rgb = Image.open(path)
    sample_image = np.array(sample_image_rgb.convert('L'))
    if sample_image.shape[0]%2 == 1:
        sample_image = sample_image[:-1]
    if sample_image.shape[1]%2 == 1:
        sample_image = sample_image[:,:-1]
    return sample_image

def piecewise_smooth_function(x):
    '''
    Returns y y=f(x) where f(x) is the given function at the project pdf
    '''
    y = (2 + np.cos(x))*np.abs(x)*np.sign(x-1)
    return y

def add_noise(data, epsilon = 2):
    '''
    Add noise * epsilon to the data
    Retursn noise-added-data
    '''
    noise = epsilon * np.random.randn(*data.shape)
    return noise + data

def modify_denoised_coefs(coefs):
    '''Handle some issues'''
    converted_list = [coefs[0]]
    for i in range(1, len(coefs)):
        converted_list.append(tuple(coefs[i]))
    return converted_list

def denoising_with_wavedec(data,
```

```
                        wavelet = 'db2',
                        level = 5,
                        threshold = 5,
                        image = True,
                        mode = 'soft'):
    '''
    Applies denoising scheme using wavedec function with given parameters.
    Returns coef, coef_thresholded_hard, coef_thresholded_soft, f_soft_denoised, f_hard_denoised
    '''
    coeffs = None
    coeffs_thresholded_hard = None
    coeffs_thresholded_soft = None
    f_soft_denoised = None
    f_hard_denoised = None

    if image:
        coeffs = pywt.wavedec2(data, wavelet, level = level)
        if mode == 'soft':
            coeffs_thresholded_soft = [pywt.threshold(c, threshold, mode='soft') for c in coeffs]
            converted_list_soft = modify_denoised_coefs(coeffs_thresholded_soft)
            f_soft_denoised = pywt.waverec2(converted_list_soft, wavelet)
        elif mode == 'hard':
            coeffs_thresholded_hard = [pywt.threshold(c, threshold, mode='hard') for c in coeffs]
            converted_list_hard = modify_denoised_coefs(coeffs_thresholded_hard)
            f_hard_denoised = pywt.waverec2(converted_list_hard, wavelet)

        elif mode == 'both':
            coeffs_thresholded_soft = [pywt.threshold(c, threshold, mode='soft') for c in coeffs]
            converted_list_soft = modify_denoised_coefs(coeffs_thresholded_soft)
            f_soft_denoised = pywt.waverec2(converted_list_soft, wavelet)
            coeffs_thresholded_hard = [pywt.threshold(c, threshold, mode='hard') for c in coeffs]
            converted_list_hard = modify_denoised_coefs(coeffs_thresholded_hard)
            f_hard_denoised = pywt.waverec2(converted_list_hard, wavelet)
        else:
            print('Wrong mode selection')
            return
    else:
        coeffs = pywt.wavedec(data, wavelet, level = level)
        if mode == 'soft':
            coeffs_thresholded_soft = [pywt.threshold(c, threshold, mode = 'soft') for c in coeffs]
            f_soft_denoised = pywt.waverec(coeffs_thresholded_soft, wavelet)

        elif mode == 'hard':
            coeffs_thresholded_hard = [pywt.threshold(c, threshold, mode = 'hard') for c in coeffs]
            f_hard_denoised = pywt.waverec(coeffs_thresholded_hard, wavelet)

        elif mode == 'both':
            coeffs_thresholded_soft = [pywt.threshold(c, threshold, mode = 'soft') for c in coeffs]
            f_soft_denoised = pywt.waverec(coeffs_thresholded_soft, wavelet)
            coeffs_thresholded_hard = [pywt.threshold(c, threshold, mode = 'hard') for c in coeffs]
```

```
            f_hard_denoised = pywt.waverec(coeffs_thresholded_hard, wavelet)

    return coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft, f_soft_denoised, f_hard_denoised

def threshold_swt(original, copy, threshold, mode):
    '''
    Applies thresholding for swt function
    '''
    for key in original[0]:
        copy[0][key] = pywt.threshold(original[0][key], value = threshold, mode=mode)
    return copy

def denoising_with_swt(data,
                       wavelet = 'db2',
                       threshold = 20,
                       level = 1,
                       start_level = 0,
                       mode = 'soft',
                       image = True):
    '''
    Applies denoising scheme using wavedec function with given parameters.
    Returns denoised_image_array_soft, denoised_image_array_hard
    '''
    denoised_image_array_soft = None
    denoised_image_array_hard = None

    if image:
        coeffs = pywt.swtn(data, wavelet = wavelet, level = level, start_level = start_level)
        converted = coeffs.copy()

        if mode == 'soft':
            converted_list_soft = threshold_swt(coeffs, converted, threshold, mode = 'soft')
            denoised_image_array_soft = pywt.iswtn(converted_list_soft, wavelet)
        elif mode == 'hard':
            converted_list_hard = threshold_swt(coeffs, converted, threshold, mode = 'hard')
            denoised_image_array_hard = pywt.iswtn(converted_list_hard, wavelet)
        elif mode == 'both':
            converted_list_soft = threshold_swt(coeffs, converted, threshold, mode = 'soft')
            denoised_image_array_soft = pywt.iswtn(converted_list_soft, wavelet)
            converted_list_hard = threshold_swt(coeffs, converted, threshold, mode = 'hard')
            denoised_image_array_soft = pywt.iswtn(converted_list_hard, wavelet)
        else:
            print('Wrong mode!')
    else:
        coeffs = pywt.swt(data, wavelet = wavelet, level = level)
        if mode == 'soft':
            coeffs_thresholded_soft = pywt.threshold(coeffs, threshold, mode='soft')
            coeffs_thresholded_soft = [tuple(coeffs_thresholded_soft[0])]
            denoised_image_array_soft = pywt.iswt(coeffs_thresholded_soft, wavelet = wavelet)
```

```
        elif mode == 'hard':
            coeffs_thresholded_hard = pywt.threshold(coeffs, threshold, mode='hard')
            coeffs_thresholded_hard = [tuple(coeffs_thresholded_hard[0])]
            denoised_image_array_hard = pywt.iswt(coeffs_thresholded_hard, wavelet = wavelet)

        elif mode == 'both':
            coeffs_thresholded_soft = pywt.threshold(coeffs, threshold, mode='soft')
            coeffs_thresholded_soft = [tuple(coeffs_thresholded_soft[0])]
            coeffs_thresholded_hard = pywt.threshold(coeffs, threshold, mode='hard')
            coeffs_thresholded_hard = [tuple(coeffs_thresholded_hard[0])]
            denoised_image_array_hard = pywt.iswt(coeffs_thresholded_hard, wavelet = wavelet)
            denoised_image_array_soft = pywt.iswt(coeffs_thresholded_soft, wavelet = wavelet)
        else:
            print('Wrong mode!')

    return denoised_image_array_soft, denoised_image_array_hard

def plot_signals(x, original, noisy, soft_denoised, hard_denoised,
                 threshold = 'NA',
                 wavelet = 'NA',
                 epsilon = 'NA',
                 mse_soft = 'NA',
                 mse_hard = 'NA'):
    '''
    Plot original signal, noisy signal and denoised signal
    '''
    if mse_soft != 'NA':
        mse_soft = round(mse_soft,4)
    if mse_hard != 'NA':
        mse_hard = round(mse_hard,4)

    plt.figure(figsize=(12, 6))

    plt.subplot(2, 2, 1)
    plt.plot(x, original)
    plt.title('Original Signal')

    plt.subplot(2, 2, 2)
    plt.plot(x, noisy)
    plt.title('Noisy Signal')

    plt.subplot(2, 2, 3)
    plt.plot(x, soft_denoised, label = 'Denoised')
    plt.plot(x, original, 'r', label = 'Truth')
    plt.legend()
    plt.title(f'Soft Thresholding Denoised Signal\nMSE: {mse_soft}')

    plt.subplot(2, 2, 4)
    plt.plot(x, hard_denoised, label = 'Denoised')
    plt.plot(x, original, 'r', label = 'Truth')
```

```
    plt.legend()
    plt.title(f'Hard Thresholding Denoised Signal\nMSE: {mse_hard}')

    plt.suptitle(f'Denoisig with Noise {epsilon}: Wavelet = {wavelet} - Threshold = {threshold}')
    plt.tight_layout()
    plt.show()

def plot_images(original, noisy = None, denoised_soft = None, denoised_hard = None):
    '''
    Plot original image, noisy image, and denoised images
    '''
    if type(noisy) == type(None):
        noisy = np.zeros(original.shape)
    if type(denoised_soft) == type(None):
        denoised_soft = np.zeros(original.shape)
    if type(denoised_hard) == type(None):
        denoised_hard = np.zeros(original.shape)

    # Display or save the original, noisy, and denoised images
    plt.figure(figsize=(14, 10))

    plt.subplot(2, 2, 1)
    plt.imshow(original, cmap = 'gray')
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(2, 3, 2)
    plt.imshow(noisy, cmap='gray')
    plt.title('Noisy Image')
    plt.axis('off')

    plt.subplot(2, 2, 3)
    plt.imshow(denoised_soft, cmap='gray')
    plt.title('Denoised Image - Mode: Soft')
    plt.axis('off')

    plt.subplot(2, 3, 5)
    plt.imshow(denoised_hard, cmap='gray')
    plt.title('Denoised Image - Mode: Hard')
    plt.axis('off')
    plt.show()

def calculate_snr(original_image, denoised_image):
    '''
    Inputs original data and denoised data
    Returns signal-to-noise ratio
    '''
    original_array = np.array(original_image).astype(float)
    denoised_array = np.array(denoised_image).astype(float)
```

```
    signal_power = np.sum(original_array ** 2)
    noise_power = np.sum((original_array - denoised_array) ** 2)

    snr = 10 * np.log10(signal_power / noise_power)
    return snr

def create_mask(image, random = True, mask_size = (512,512), true_count = 50544):
    '''
    Create a mask for the condition given at the project pdf or randomly allocated mask
    '''
    if random:
        mask = np.zeros(mask_size, dtype=bool)
        indices = np.random.choice(np.prod(mask_size), true_count, replace=False)
        mask.flat[indices] = True
    else:
        mask = np.zeros_like(image)
        mask[::10, :] = 1
        mask[:, ::10] = 1
        mask = mask > 0
    return mask


def wavelet_inpainting(A, mask, B0, max_iter, wavelet = 'db2', level = 5, threshold = 20):
    '''
    Applies inpainting scheme described at the pdf using inputed parameters.
    Inpaints with both wavedec and swt, and with both hard and soft thresholding.
    Returns denoised image: wavedec_soft, wavedec_hard, swt_soft, swt_hard
    '''
    A[mask] = B0
    A_wavedec_soft = A.copy()
    A_wavedec_hard = A.copy()
    A_swt_soft = A.copy()
    A_swt_hard = A.copy()


    for _ in range(max_iter):
        _, _, _, image_wavedec_soft, _ = denoising_with_wavedec(A_wavedec_soft,
                                                                wavelet = wavelet,
                                                                level = level,
                                                                threshold = threshold,
                                                                mode = 'soft',
                                                                image = True)

        _, _, _, _, image_wavedec_hard = denoising_with_wavedec(A_wavedec_hard,
                                                                wavelet = wavelet,
                                                                level = level,
                                                                threshold = threshold,
                                                                mode = 'hard',
                                                                image = True)
```

```
        image_swt_soft, _ = denoising_with_swt(A_swt_soft, wavelet = wavelet,
                                               threshold = threshold,
                                               level = level,
                                               start_level = 0,
                                               mode = 'soft',
                                               image = True)

        _, image_swt_hard = denoising_with_swt(A_swt_hard, wavelet = wavelet,
                                               threshold = threshold,
                                               level = level,
                                               start_level = 0,
                                               mode = 'hard',
                                               image = True)

        A_wavedec_soft[mask] = image_wavedec_soft[mask]
        A_wavedec_hard[mask] = image_wavedec_hard[mask]
        A_swt_soft[mask] = image_swt_soft[mask]
        A_swt_hard[mask] = image_swt_hard[mask]

    return A_wavedec_soft, A_wavedec_hard, A_swt_soft, A_swt_hard
```

Section 2

```
### 2.1 A Univariate Functions with Noise
## Q2.1
# Generate Function and Data Points
N = 2**10
x = np.linspace(-2, 2, N)
f_vector = piecewise_smooth_function(x)

# Apply DWT, returns coefficeints
wavelet_type = 'db2'
level = 2
coeffs, _, _, _, _ = denoising_with_wavedec(f_vector,
                                            wavelet = wavelet_type,
                                            level = level,
                                            image = False)

# Plot the coefficients and the original signal
plt.figure(figsize=(10,8))
plt.subplot(4, 1, 1)

plt.plot(x, f_vector)
plt.title('Original Signal')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')
for i, coef in enumerate(coeffs):
    plt.subplot(4, 1, i+2)
    if i == 0:
        plt.semilogy(np.arange(len(coef)), np.abs(coef), label=f'Scaling Level {len(coeffs)-1}')
        plt.title(f'Scaling Function (Log-Scale) - Wavelet: {pywt.Wavelet(wavelet_type).name}')
        plt.xlabel('Coefficient Index')
    else:
        plt.semilogy(np.arange(len(coef)), np.abs(coef), label=f'Level {len(coeffs)-i}')
        plt.title(f'Wavelet Coefficients (Log-Scale) - Wavelet: {pywt.Wavelet(wavelet_type).name}')
        plt.xlabel('Coefficient Index')
    plt.ylabel('Log-Scale Amplitude')
    plt.legend()

plt.tight_layout()
plt.show()

## Q2.2
# Add noise to the signal
epsilon = 2
f_noisy = add_noise(f_vector, epsilon = epsilon)

# Denoising with DWT, random combination
wavelet_type = 'db2'
level = 8
threshold = 1
coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
```

```
f_soft_denoised, f_hard_denoised = denoising_with_wavedec(f_noisy, wavelet = wavelet_type, level = level,
threshold = threshold, mode = 'both',image = False)

plot_signals(x, f_vector, f_noisy, f_soft_denoised, f_hard_denoised,
threshold = threshold, wavelet = wavelet_type, epsilon = epsilon)



## Q2.3

# Denoising with DWT, random combination + MSE added
wavelet_type = 'db2'
level = 8
threshold = 1

coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
f_soft_denoised, f_hard_denoised = denoising_with_wavedec(f_noisy, wavelet = wavelet_type, level = level,
threshold = threshold,mode = 'both',image = False)

mse_soft = mean_squared_error(f_vector, f_soft_denoised)
mse_hard = mean_squared_error(f_vector, f_hard_denoised)

plot_signals(x, f_vector, f_noisy, f_soft_denoised, f_hard_denoised, threshold = threshold,
             wavelet = wavelet_type, epsilon = epsilon, mse_soft = mse_soft, mse_hard = mse_hard)


# Hyperparameter calculation using grid-search like algo.
level_list = np.linspace(0,8,9, dtype=int)
threshold_list = np.linspace(0,10, 21)
mse_limit = 100000
wavelet_type = 'db2'
result_val = []

for lev in level_list:
    for thresh in threshold_list:
        coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
        f_soft_denoised, f_hard_denoised = denoising_with_wavedec(f_noisy,wavelet = wavelet_type,
        level = lev, threshold = thresh,mode = 'both',image = False)

        mse_soft = mean_squared_error(f_vector, f_soft_denoised)
        mse_hard = mean_squared_error(f_vector, f_hard_denoised)
        if mse_soft < mse_limit:
            mse_limit = mse_soft
            result_val = [lev, thresh, mse_limit]
        if mse_hard < mse_limit:
            mse_limit = mse_hard
            result_val = [lev, thresh, mse_limit]


# Combination that minimises MSE is selected as hyperparameter and DWT is applied.
```

```
coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
f_soft_denoised, f_hard_denoised = denoising_with_wavedec(f_noisy, wavelet = wavelet_type,
level = result_val[0],
threshold = result_val[1], mode = 'both',image = False)
mse_soft = mean_squared_error(f_vector, f_soft_denoised)
mse_hard = mean_squared_error(f_vector, f_hard_denoised)

plot_signals(x, f_vector, f_noisy, f_soft_denoised, f_hard_denoised, threshold = result_val[1],
             wavelet = wavelet_type, epsilon = epsilon, mse_soft = mse_soft, mse_hard = mse_hard)

### Images with Noise
## Q2.4

# Image is loaded
image_path = 'sample_image.png'
original_image = load_image(image_path)

# Noise is added
epsilon = 20
noisy_image = add_noise(original_image, epsilon = epsilon)

## Q2.5
# Denoising the image with DWT, random combination
level = 3
threshold = 20
wavelet = 'db2'

coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
denoised_image_soft, denoised_image_hard = denoising_with_wavedec(
    noisy_image,
    wavelet = wavelet,
    level = level,
    threshold = threshold,
    mode = 'both',
    image = True)

# Plot original image, noisy image, denoised with soft thresholding and denoised with hard thresholding
plot_images(original_image, noisy_image, denoised_image_soft, denoised_image_hard)

# calculate and output snr value
snr_value = calculate_snr(original_image, denoised_image_soft)
print(snr_value)

# Hyperparameter calculation using grid-search like algo.
level_list = [1,2,3,4,5,6]
threshold_list = np.linspace(5,100,20)
wavelet_list = ['haar', 'db2', 'bior2.2', 'sym4', 'coif2']
result_list = []
result_wave = []
```

```
for wave in wavelet_list:
    for lev in level_list:
        for thresh in threshold_list:
            coeffs, coeffs_thresholded_hard, coeffs_thresholded_soft,
            denoised_image_soft, denoised_image_hard = denoising_with_wavedec(
                noisy_image,
                wavelet = wave,
                level = lev,
                threshold = thresh,
                mode = 'both',
                image = True)

            snr_value_soft = calculate_snr(original_image, denoised_image_soft)
            snr_value_hard = calculate_snr(original_image, denoised_image_hard)
            result_list.append([lev, thresh, snr_value_soft, snr_value_hard])
            result_wave.append(wave)


### Using a Redundant Wavelet Transform
## Q2.7

# redundant wavelet transformation (SWT) is applied
wavelet = ['db2', 'sym4', 'coif2'][0]
denoised_image_array_soft, denoised_image_array_hard = denoising_with_swt(noisy_image,
                                                                          wavelet = wavelet,
                                                                          threshold = 20,
                                                                          level = 3,
                                                                          start_level = 0,
                                                                          mode = 'both',
                                                                          image = True)


# Hyperparameter calculation using grid-search like algo.
level_list = [1,2,3,4,5,6]
threshold_list = np.linspace(5,100,20, dtype='int')
wavelet_list = ['db2', 'sym4', 'coif2']
result_list_swt = []
result_wave_swt = []

for wave in wavelet_list:
    for lev in level_list:
        for thresh in threshold_list:
            denoised_image_array_soft, denoised_image_array_hard = denoising_with_swt(noisy_image,
                                                                                      wavelet = wave,
                                                                                      threshold = thresh,
                                                                                      level = lev,
                                                                                      start_level = 0)

            snr_value_soft = calculate_snr(original_image, denoised_image_array_soft)
            snr_value_hard = calculate_snr(original_image, denoised_image_array_hard)
```

```
            result_list_swt.append([lev, thresh, snr_value_soft, snr_value_hard])
            result_wave_swt.append(wave)



## Q2.8

# SWT is applied to the problem at Q2.2
epsilon = 2
wavelet_type = 'db2'
level = 3
start_level = 0
threshold = 4.5

f_noisy = add_noise(f_vector, epsilon=epsilon)

f_soft_denoised, f_hard_denoised = denoising_with_swt(f_noisy,
                                                      wavelet = wavelet_type,
                                                      threshold = threshold,
                                                      level = level,
                                                      start_level = start_level,
                                                      mode = 'both',
                                                      image = False)
mse_soft = mean_squared_error(f_vector, f_soft_denoised)
mse_hard = mean_squared_error(f_vector, f_hard_denoised)

plot_signals(x, f_vector, f_noisy, f_soft_denoised, f_hard_denoised,
threshold, wavelet_type, epsilon, mse_soft, mse_hard)


# Hyperparameter calculation using grid-search like algo.
level_list = np.linspace(1,8,8, dtype=int)
threshold_list = np.linspace(0,10, 21)[1:]
mse_limit = 100000
wavelet_type = 'db2'
result_val = []

for lev in level_list:
    for thresh in threshold_list:

        f_soft_denoised, f_hard_denoised = denoising_with_swt(f_noisy,
                                                      wavelet = wavelet_type,
                                                      threshold = thresh,
                                                      level = lev,
                                                      mode = 'both',
                                                      image = False)
        mse_soft = mean_squared_error(f_vector, f_soft_denoised)
        mse_hard = mean_squared_error(f_vector, f_hard_denoised)
        if mse_soft < mse_limit:
            mse_limit = mse_soft
```

```
            result_val = [lev, thresh, mse_limit]
        if mse_hard < mse_limit:
            mse_limit = mse_hard
            result_val = [lev, thresh, mse_limit]

# Best combination is outputted.
print(result_val)

## Q2.9
# image is loaded
image_with_line = original_image.copy()

# lines are added
image_with_line[0::10,:] = 0
image_with_line[:,0::10] = 0

# swt is applied
wavelet = 'coif2'
level = 5
start_level = 1
threshold = 50
denoised_image_soft, denoised_image_hard = denoising_with_swt(image_with_line,
                                                              wavelet = wavelet,
                                                              threshold = threshold,
                                                              level = level,
                                                              start_level = start_level,
                                                              image = True,
                                                              mode = 'both')

plot_images(original_image, image_with_line, denoised_image_soft, denoised_image_hard)

# dwt is applied
wavelet = 'coif2'
level = 5
threshold = 50
_, _, _, denoised_image_soft, denoised_image_hard = denoising_with_wavedec(image_with_line,
                                                              wavelet =  wavelet,
                                                              threshold = threshold,
                                                              level = level,
                                                              image = True,
                                                              mode = 'both')
plot_images(original_image, image_with_line, denoised_image_soft, denoised_image_hard)
```

Section 3

```
##### 3 - Wavelet-based Inpainting
### 3.2 Formulation of an inpainting Problem in Python

# Sample image is loaded
image_path = 'sample_image_2.tif'
original_image = load_image(image_path)


# noise added. if random = True, then random noise added, otherwise grid-like lines are added.
image_with_line = original_image.copy()
mask = create_mask(image_with_line, random = True, mask_size = original_image.shape, true_count=12636)
image_with_line[mask] = 0



### 3.3 An Iterative Algorithm
## Q3.1

# Hyperparameter calculation using grid-search like algo.
# It applies the iterative algorithm with different level,
# wavelet, threshold, wavelet scheme, and thresholding scheme, and store them.
A = image_with_line
B0 = 0

threshold_list = np.linspace(10,100,10)
level_list = [1,2,3,4,5]
max_iter = 500
level = 7
wavelet_list = ['db2', 'sym4', 'coif2']
results_all = []
results_wave = []

for wave in wavelet_list:
    for thresh in threshold_list:
        for lev in level_list:
            image_wavedec_soft, image_wavedec_hard, image_swt_soft, image_swt_hard = wavelet_inpainting(A,
            mask, B0, max_iter, wavelet = wave, level = lev, threshold = thresh)

            results_all.append([lev, thresh,     calculate_snr(original_image, image_wavedec_soft),
                calculate_snr(original_image, image_wavedec_hard),
                calculate_snr(original_image, image_swt_soft),
                calculate_snr(original_image, image_swt_hard)])
            results_wave.append(wave)

max_row, max_col = np.unravel_index(np.argmax(np.array(results_all)[:,2:]), (150,4))
hard_thresh = max_col % 2 == 1
max_wave = results_wave[max_row]
max_level, max_thresh, _, _, _, _= results_all[max_row]
max_level, max_thresh, max_wave, hard_thresh, max_col
```

```
# applies the best combination and shows the image
max_result = [image_wavedec_soft, image_wavedec_hard, image_swt_soft, image_swt_hard] = wavelet_inpainting(
    A, mask, B0, 1000,wavelet = max_wave, level = max_level, threshold = max_thresh)
Image.fromarray(np.uint8(max_result[max_col]))


# creates plot for grid-serach-like results
fig, ax = plt.subplots(figsize=(14, 6))

ax.plot(np.array(results_all)[:,2], '-', label = 'Soft Threshold - DWT', color='skyblue', linewidth=2)
ax.plot(np.array(results_all)[:,3], '--', label = 'Hard Threshold - DWT', color='skyblue', linewidth=2)
ax.plot(np.array(results_all)[:,4], '-', label = 'Soft Threshold - SWT', color='lightcoral', linewidth=2)
ax.plot(np.array(results_all)[:,5], '--', label = 'Hard Threshold - SWT', color='lightcoral', linewidth=2)
for item in np.linspace(4.5,149.5,30)[:-1]:
    ax.axvline(item, c = 'gray', lw = 0.5)

plt.axvline(49.5, c='k')
plt.axvline(99.5, c='k')

custom_xticks = [(0, 50, 'DB2'), (50, 100, 'SYM4'), (100, 150, 'COIF2')]
xtick_positions = [(start + end) / 2 for start, end, _ in custom_xticks]
xtick_labels = [label for _, _, label in custom_xticks]
ax.set_xticks(xtick_positions)
ax.set_xticklabels(xtick_labels)

fig.patch.set_facecolor('mintcream')  # Set figure background color
ax.set_facecolor('mintcream')  # Set Axes background color

ax.grid(color='white', linestyle='--', linewidth=0.5)

ax.set_ylabel('SNR')
ax.set_title('SNR Values of Different Denoising Schemes for Sample Image 1 - Grid Noise')

ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.set_axisbelow(True)
ax.xaxis.labelpad = 10
ax.yaxis.labelpad = 10
ax.tick_params(axis='both', which='both', length=0)
ax.spines['bottom'].set_linewidth(0.5)
ax.spines['left'].set_linewidth(0.5)

plt.show()
```