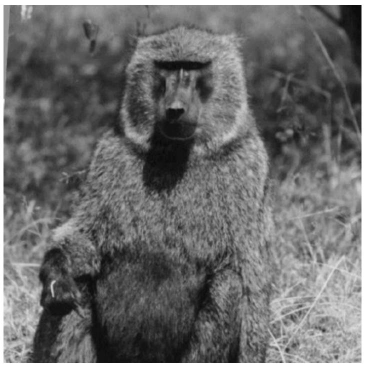
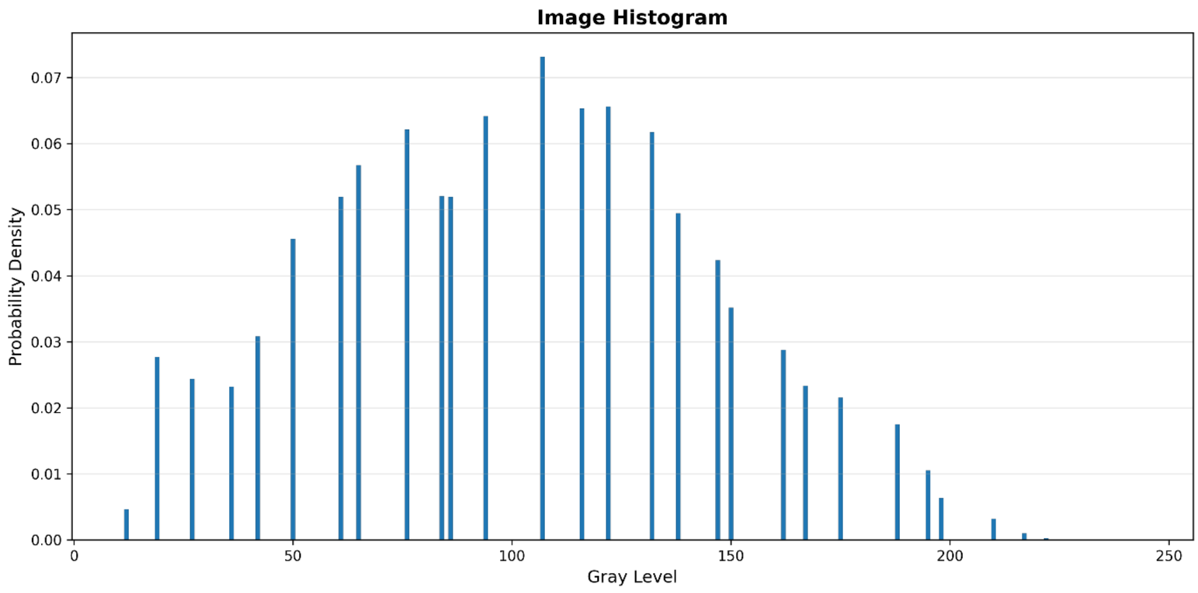
Implementation part (2)

Question 1:

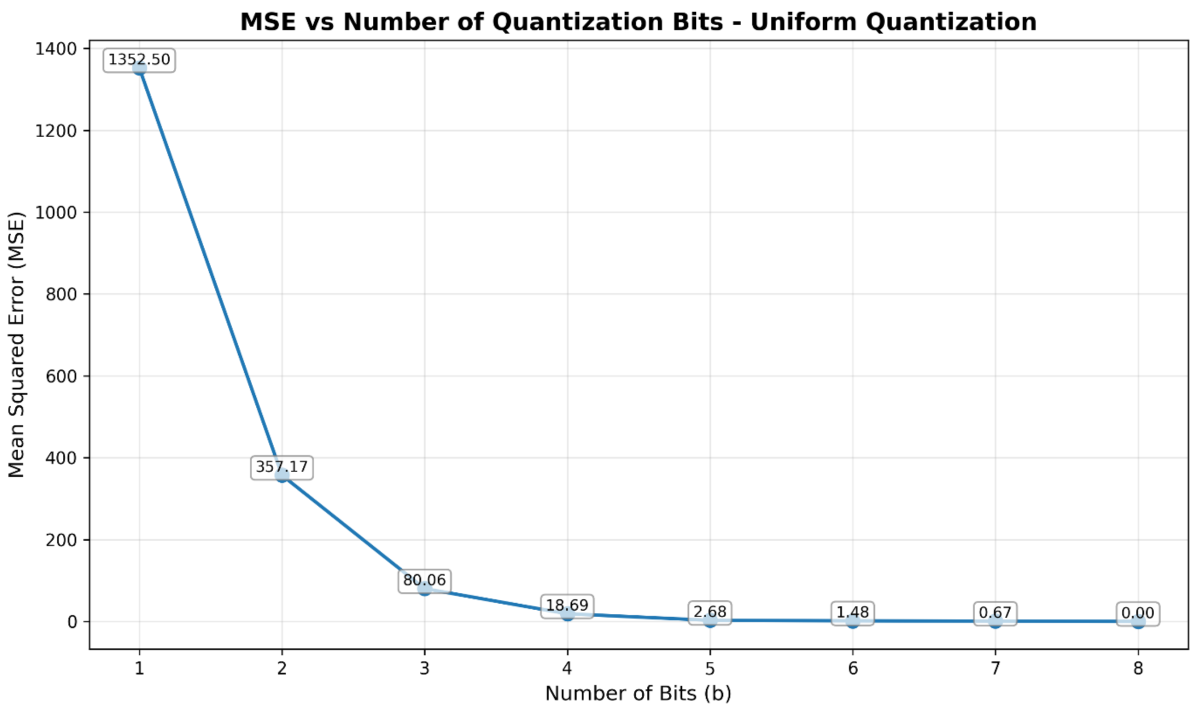
1. The 512x512 gray scale picture we choose is:



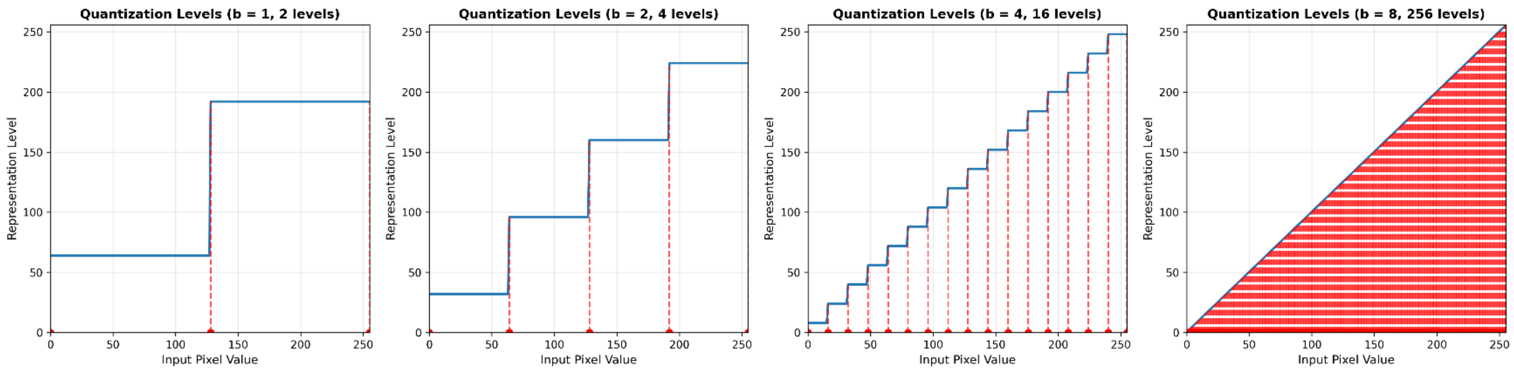
Its PDF of the gray levels is (not uniform):



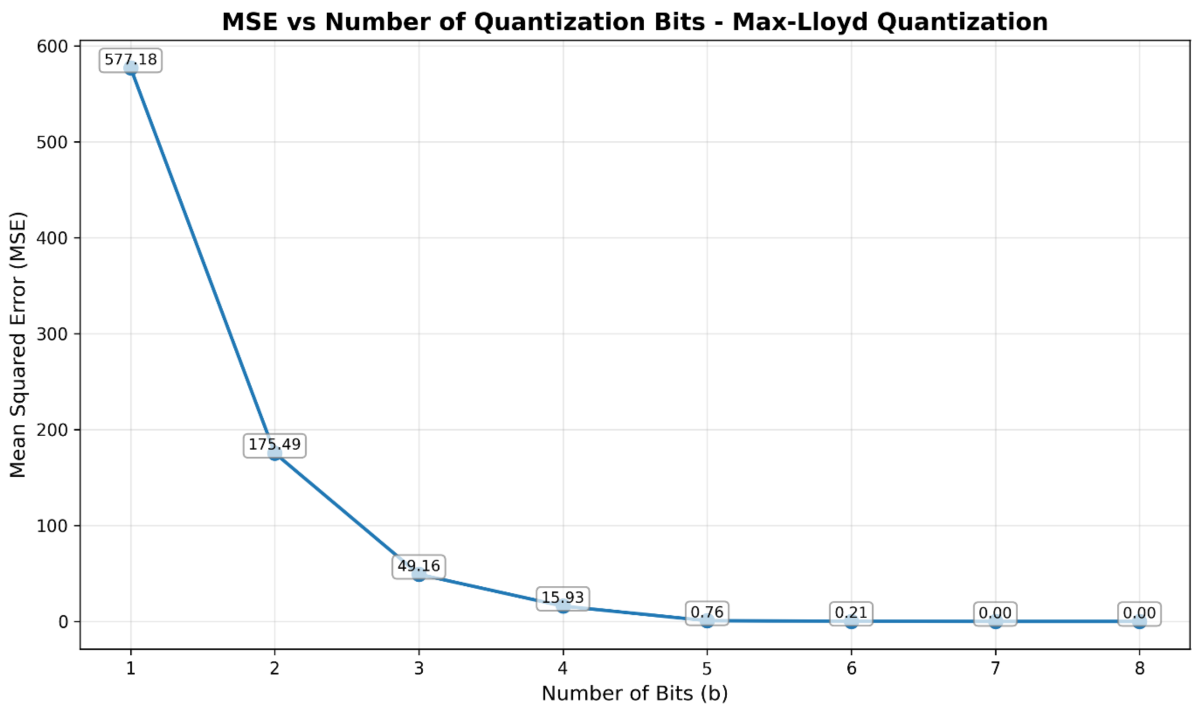
2. We applied uniform quantization on the image using b bits per pixel
    a. This is the MSE as function of bit budget b for $b = 1, \ldots, 8$ (code in quantization.py)



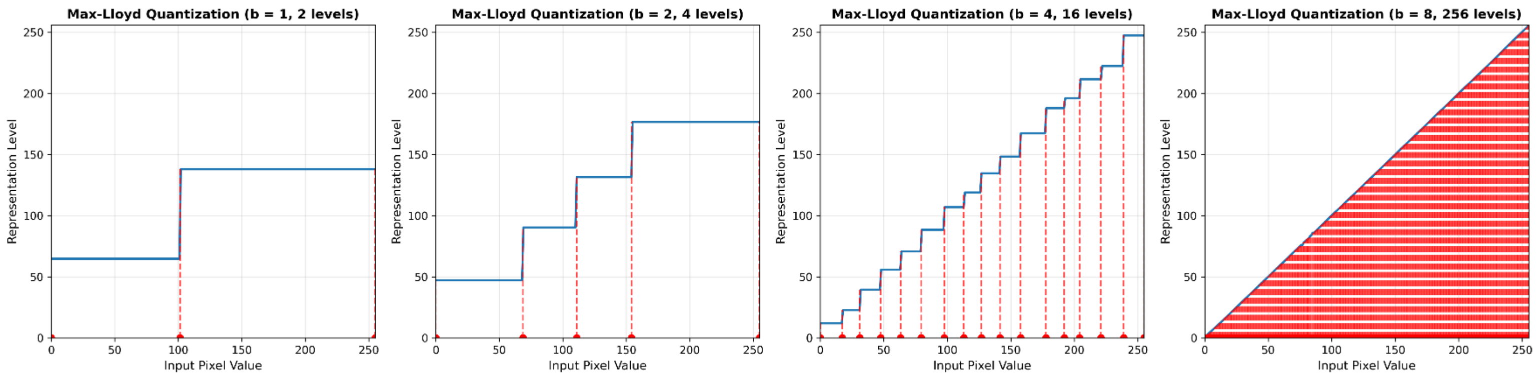
    b. Decision and representation levels for representative b values:



3. The implementation of Max-Loyd algorithm is in quantization.py file.
4. Implementation in quantization.py file
    a. This is the MSE as function of bit budget b for $b = 1, \dots, 8$ (code in quantization.py)



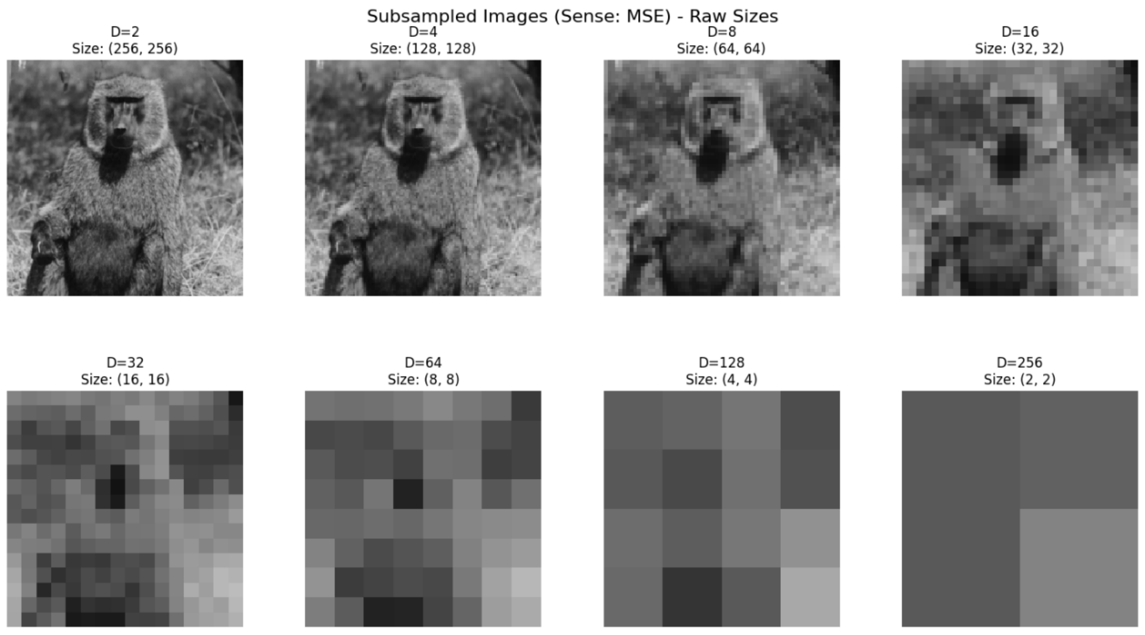
    b. Decision and representation levels for representative b values:



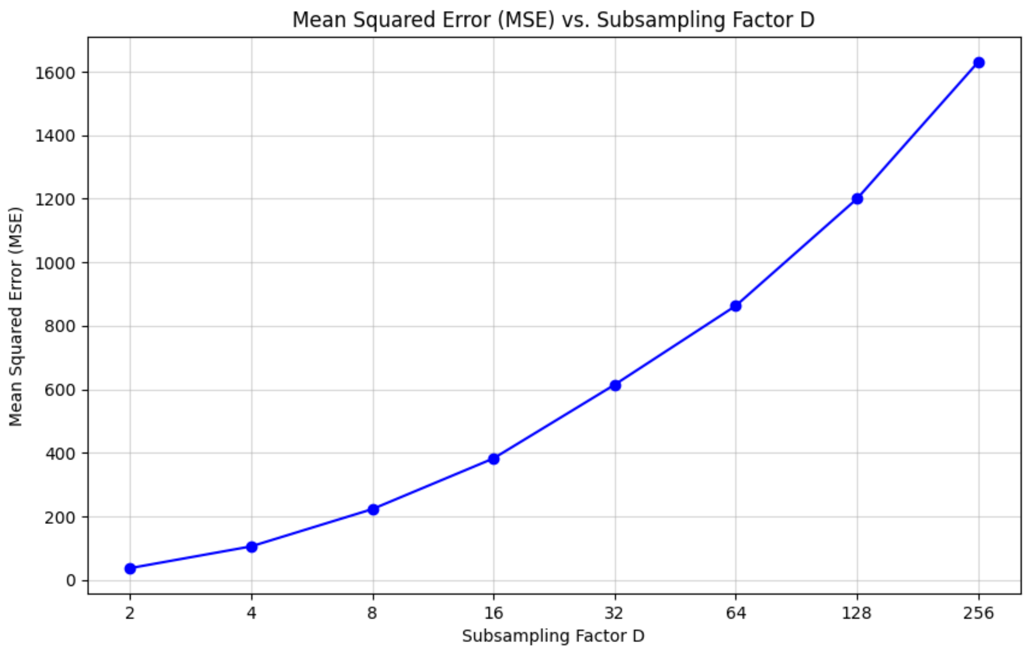
5. We observe that Max-Lloyd quantization consistently yields significantly lower MSE values than uniform quantization across all bit budgets (b) . This is expected, as uniform quantization divides the pixel range into equal intervals without regarding the specific image statistics. In contrast, the Max-Lloyd algorithm explicitly minimizes the MSE by iteratively adapting the decision and representation levels to fit the image's specific probability density function.

Question 2:

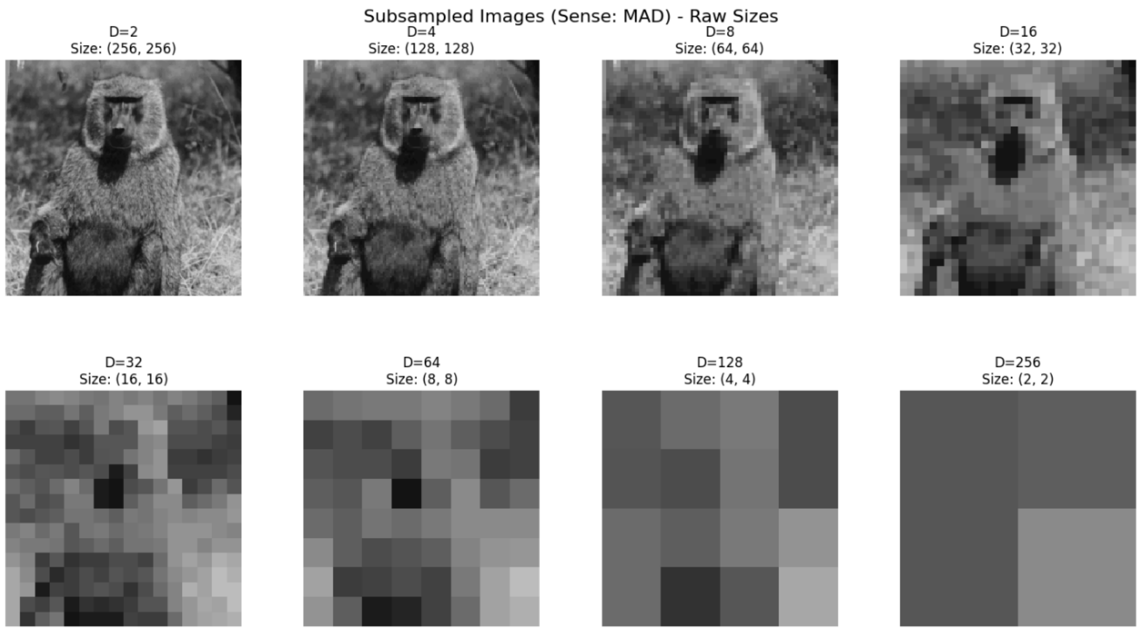
1. As proved in class, in the MSE sense, the optimal number for a sub-sample is the average of the function over the samples in the sub-sample. In the MAD sense, the optimal number for a sub-sample is the median of the function over the samples in the sub-sample.
    a. Sub-sampled images in MSE sense, for different sub-sampling factor, D:



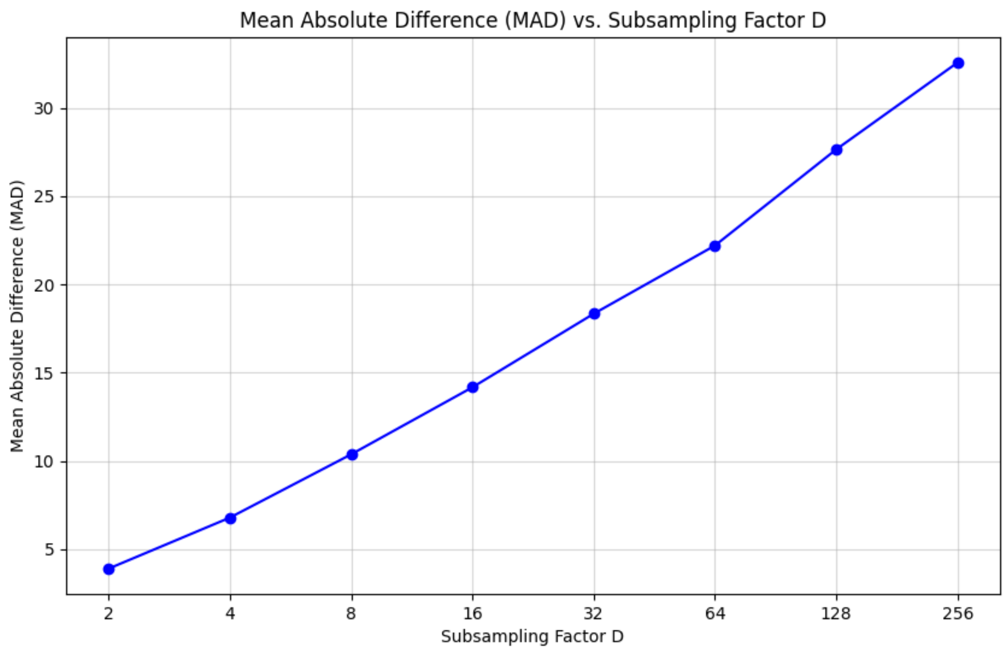
In the graph of MSE as function of Sub-sampling factor D, the horizontal axis (Sub-sampling factor D) is taken in log scale so the different values of D will be evenly spaced.



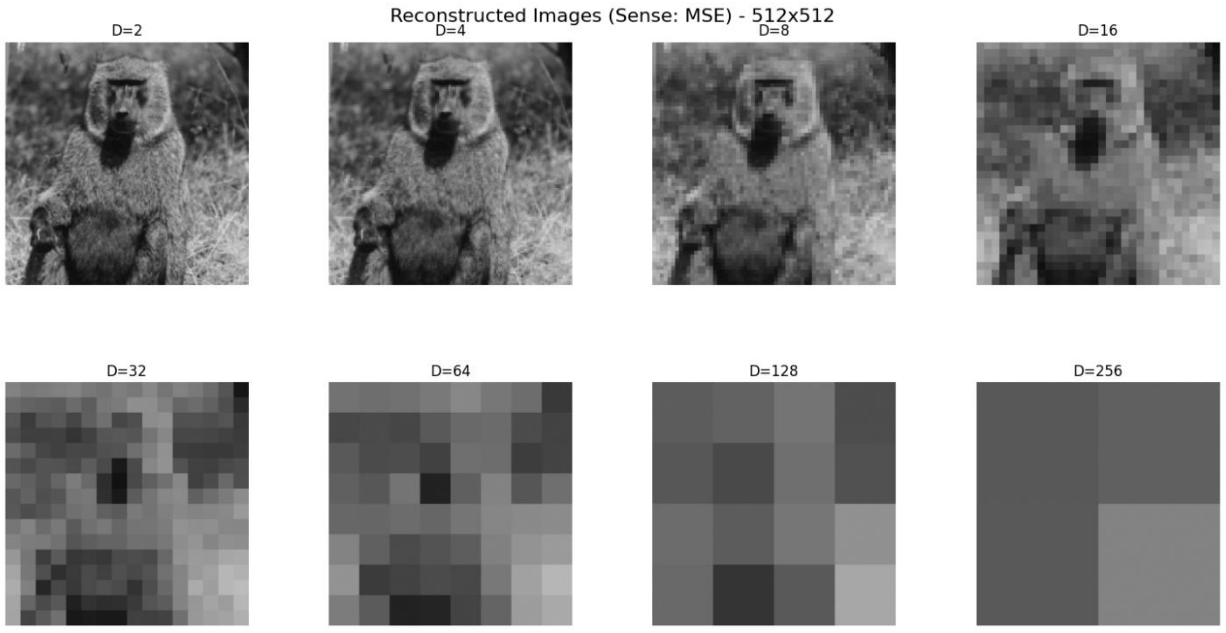
b. Sub-sampled images in MAD sense, for different sub-sampling factor, D:

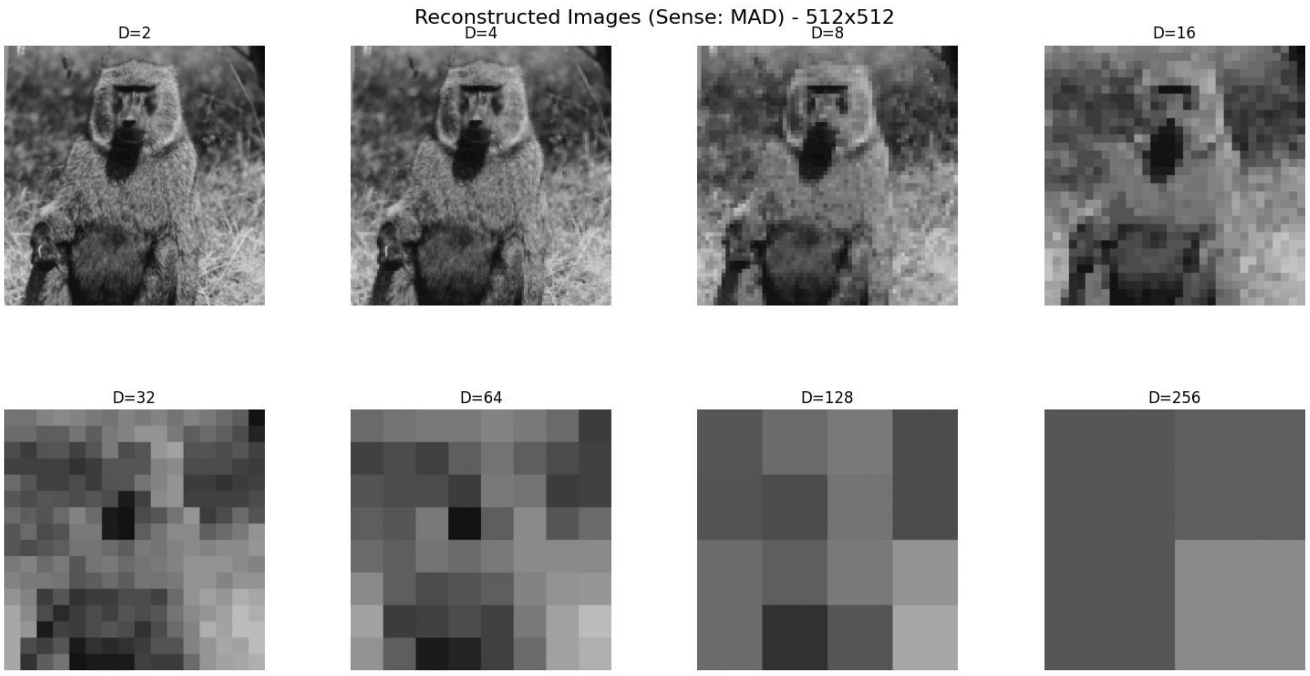


In the graph of MAD as function of Sub-sampling factor D, the horizontal axis (Sub-sampling factor D) is taken in log scale so the different values of D will be evenly spaced.



2. We reconstruct the image by taking each pixel and duplicate it number of times such that we get a 512x512 image (original size). For example, in case D=2, we get a sampled image in size 256x256. In the reconstructed image, we will take each pixel and duplicate it 4 times – meaning each 1x1 pixel becomes 2x2 four pixels, and the result is 512x512 image.

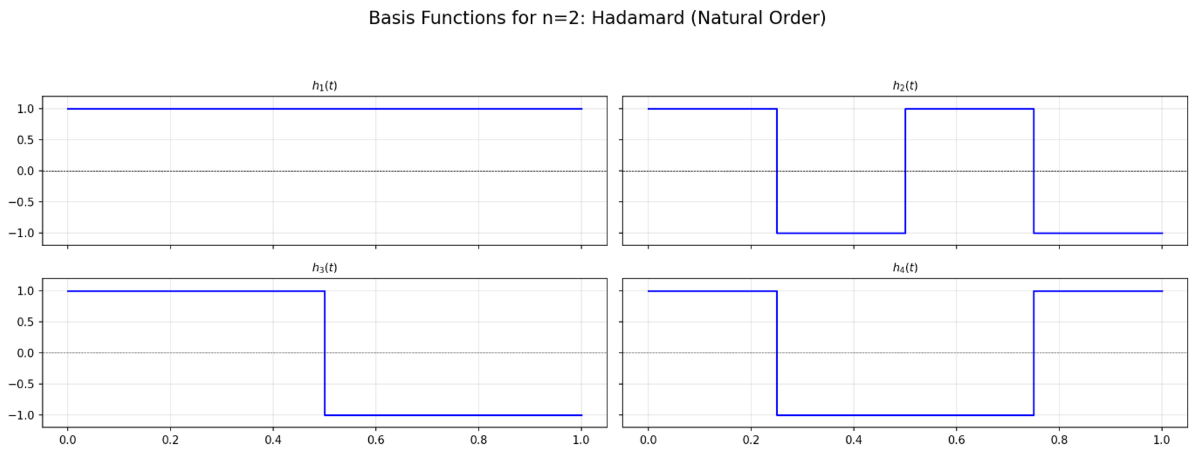


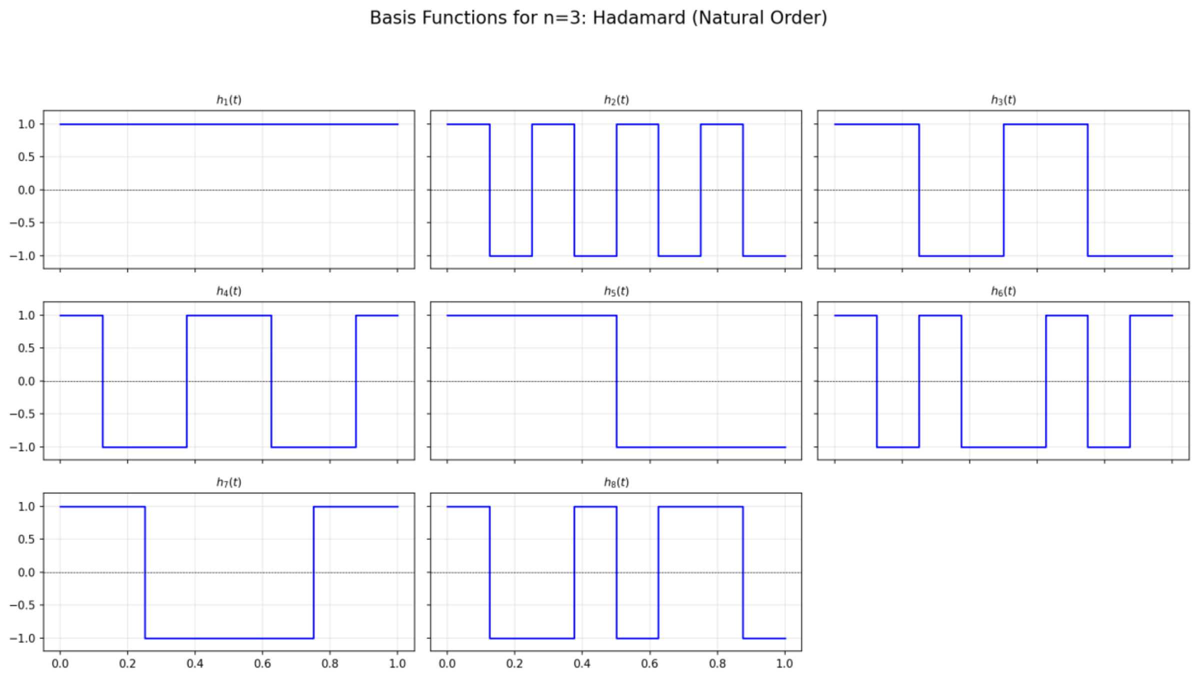


3. As the sub-sampling factor D increases, fewer pixels are retained from the original image, since each $D \times D$ block is represented by a single sample. This leads to a loss of spatial detail and high-frequency information. After reconstruction, the image becomes increasingly blurred, and the reconstruction error increases accordingly, as reflected by higher MSE and MAD values, as we can see in the graph presented in previous sections of this answer.

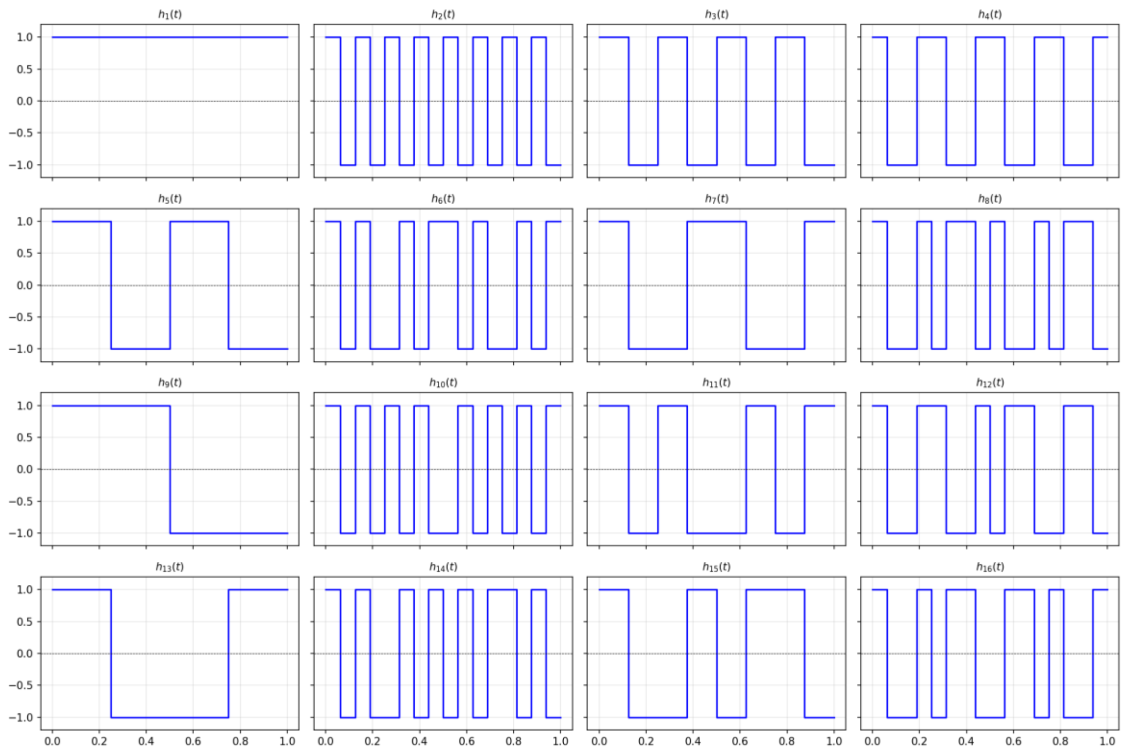
Question 3:

a. Implementation is in python file **matrices.py** by a function named **Hadamard_matrix** which get n as an input and returns the Hadamard matrix of size $2^n \times 2^n$.
b. We did it by taking the columns of the appropriate Hadamard matrix multiplied by normalization factor – square of N.

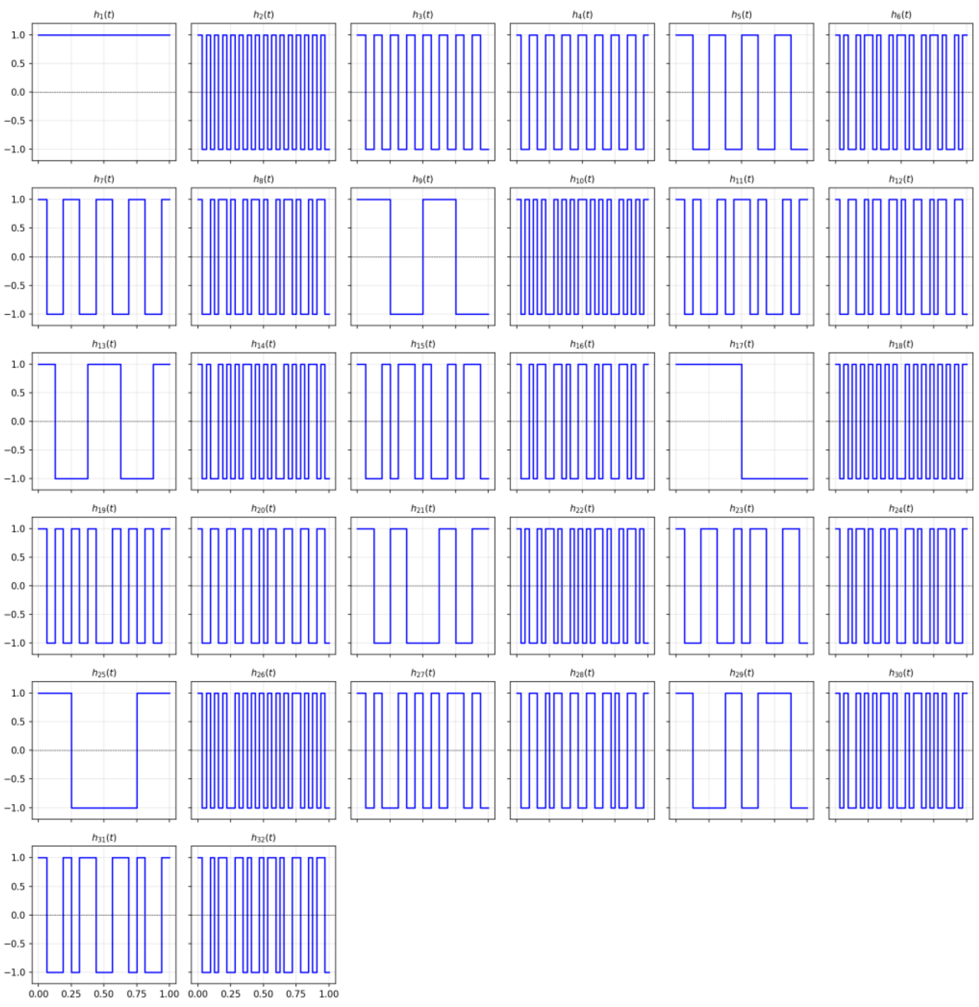




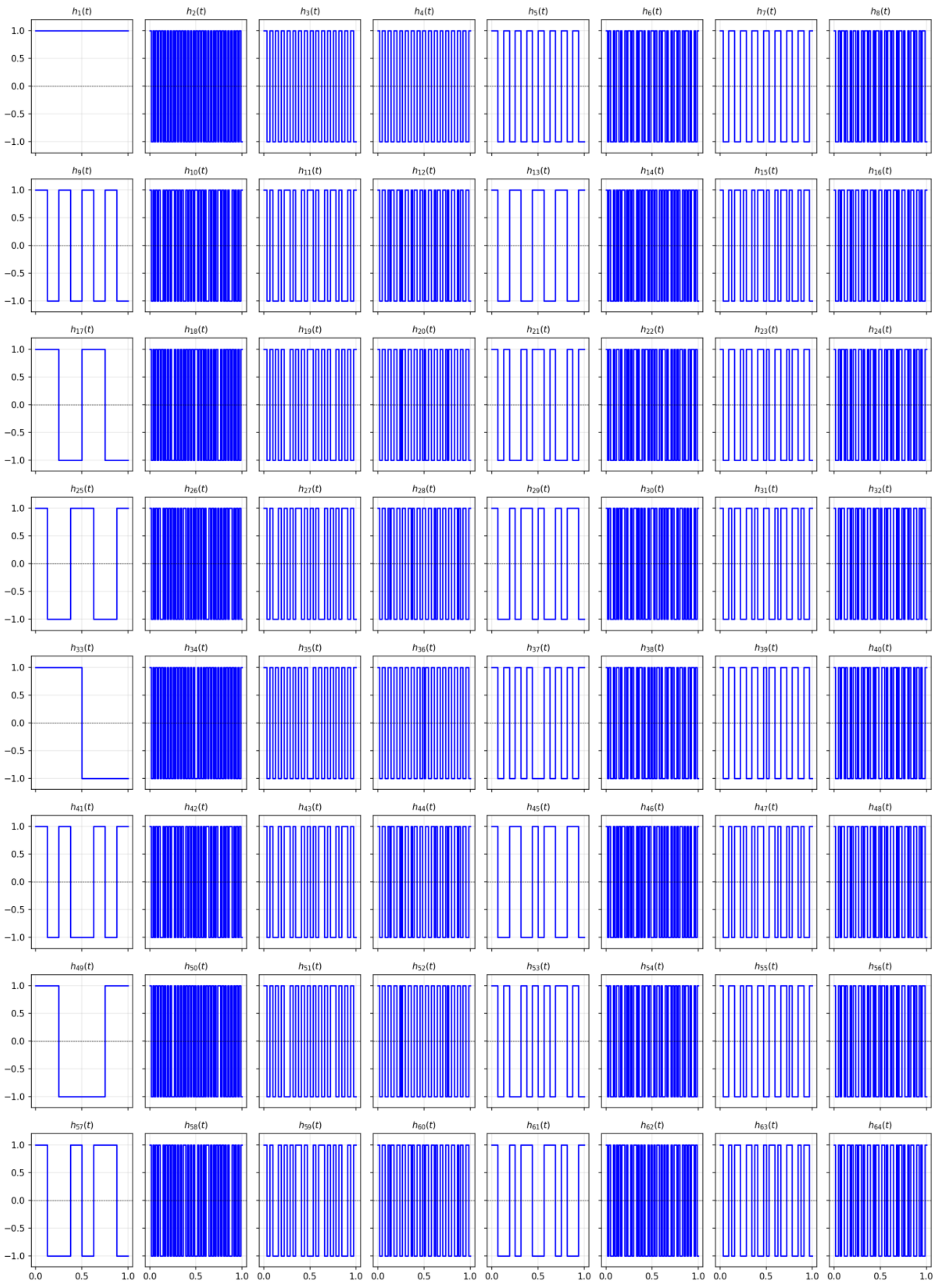
Basis Functions for n=4: Hadamard (Natural Order)



Basis Functions for n=5: Hadamard (Natural Order)

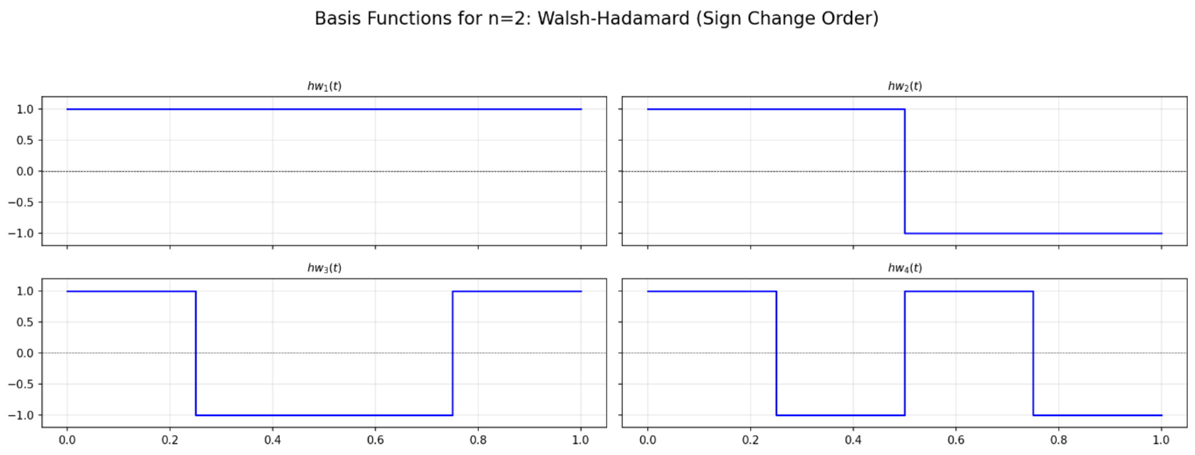


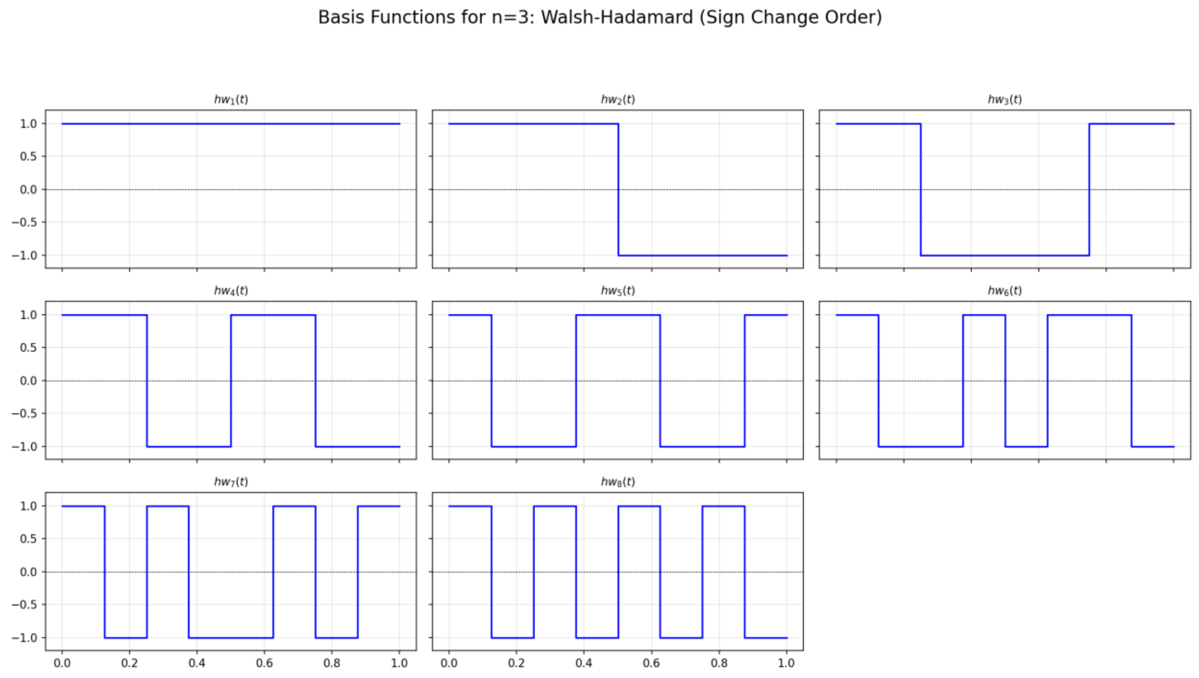
Basis Functions for n=6: Hadamard (Natural Order)



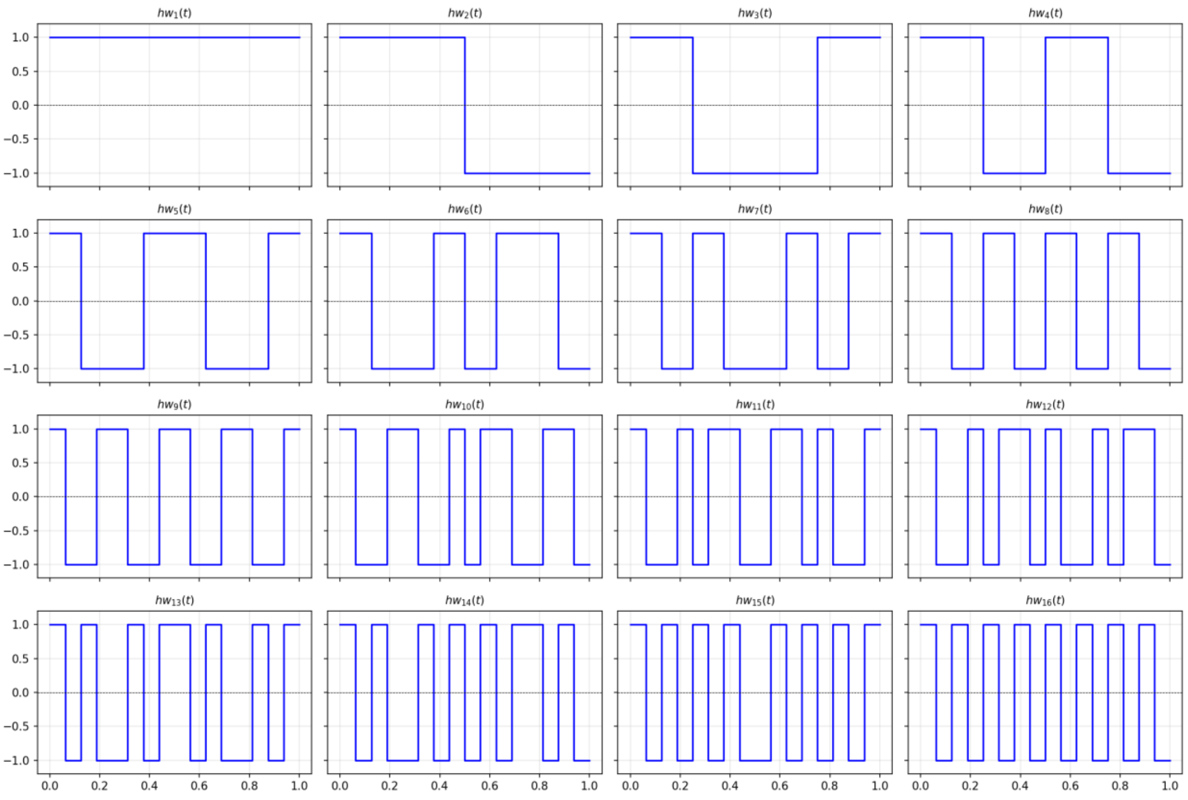
c. For Walsh-Hadamard, we took the Hadamard matrix in the desired size, and sorted its rows based on sign changes – in increasing order.

d. We did it by taking the columns of the appropriate Walsh-Hadamard matrix multiplied by normalization factor – square of N:

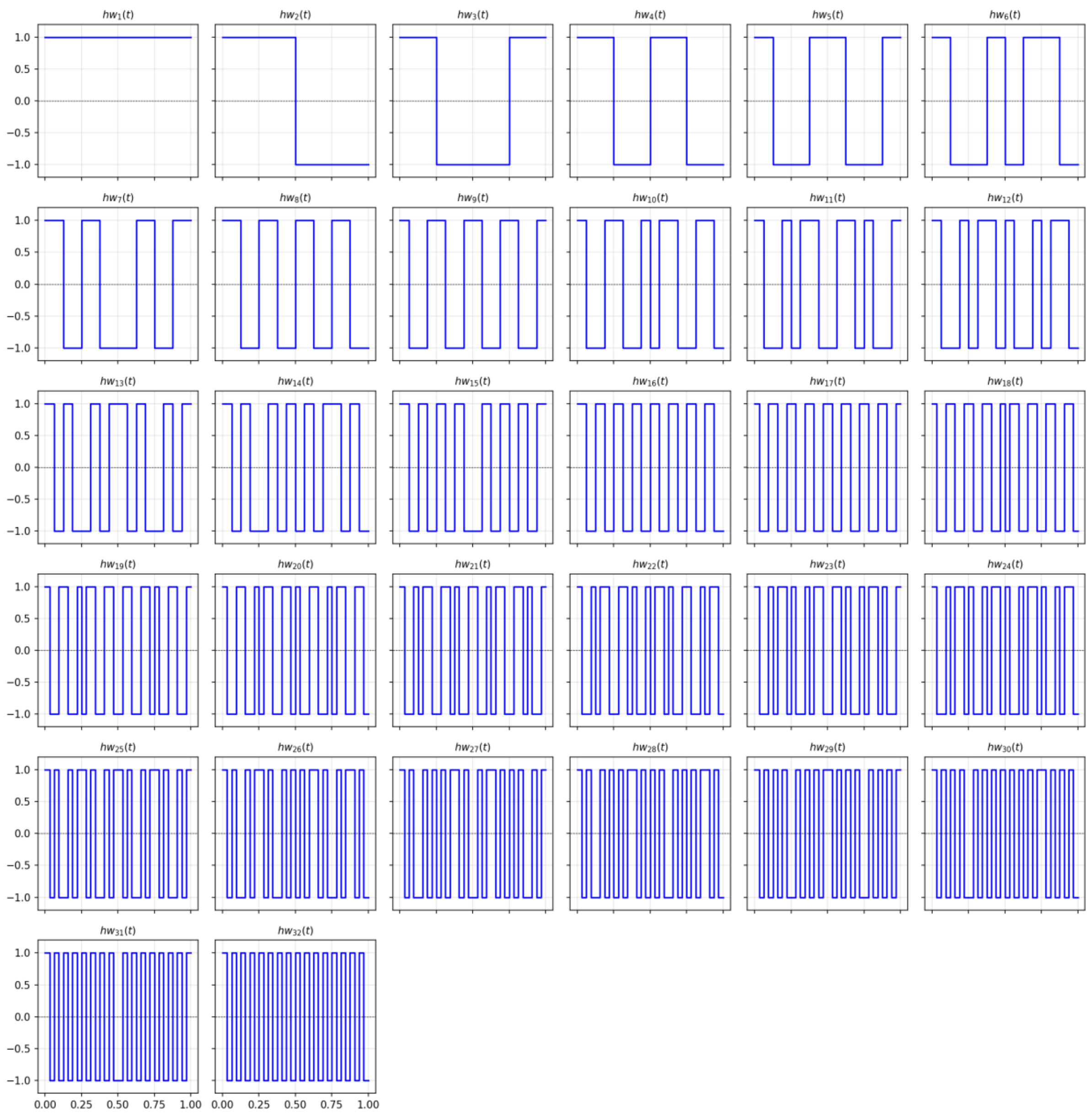




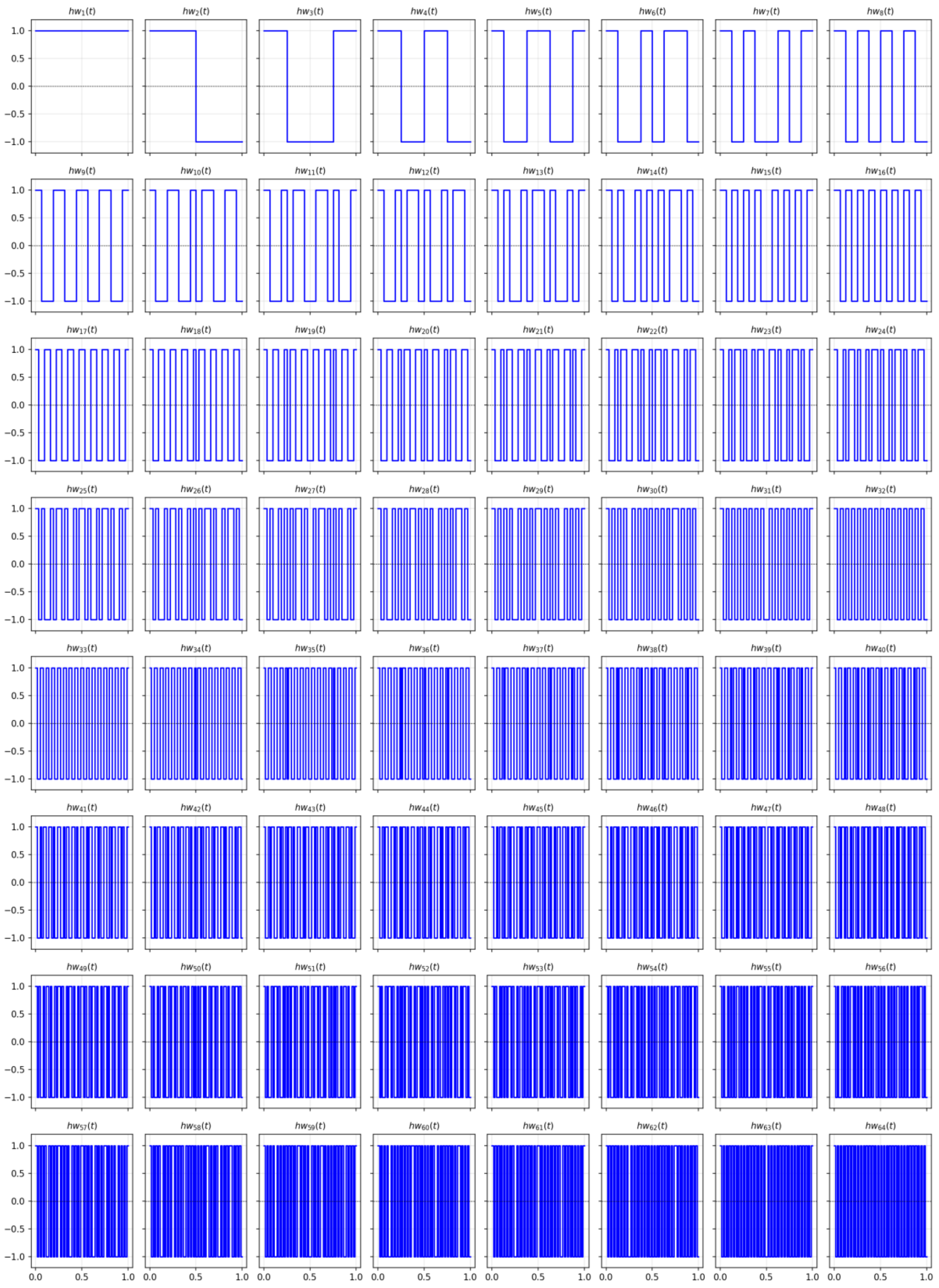
Basis Functions for n=4: Walsh-Hadamard (Sign Change Order)



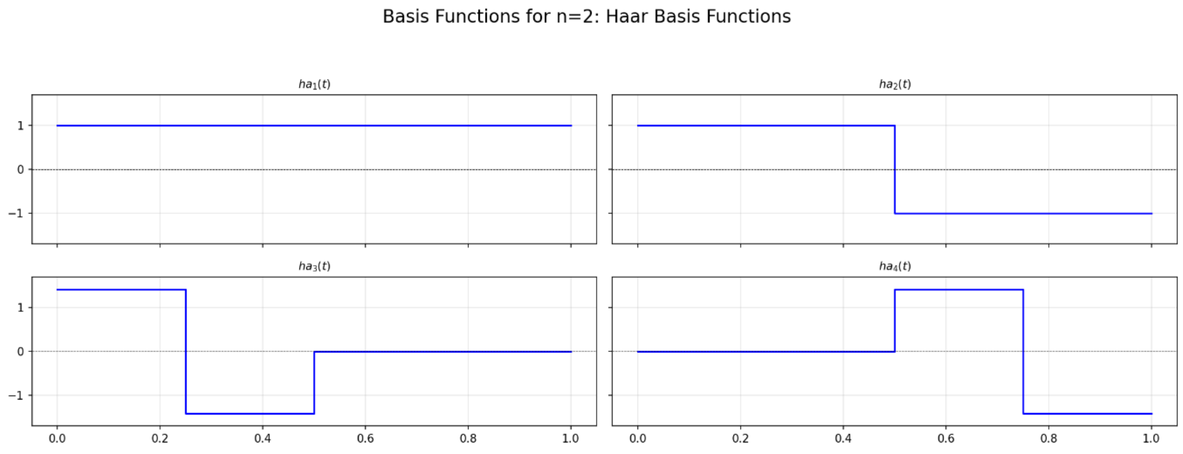
Basis Functions for n=5: Walsh-Hadamard (Sign Change Order)

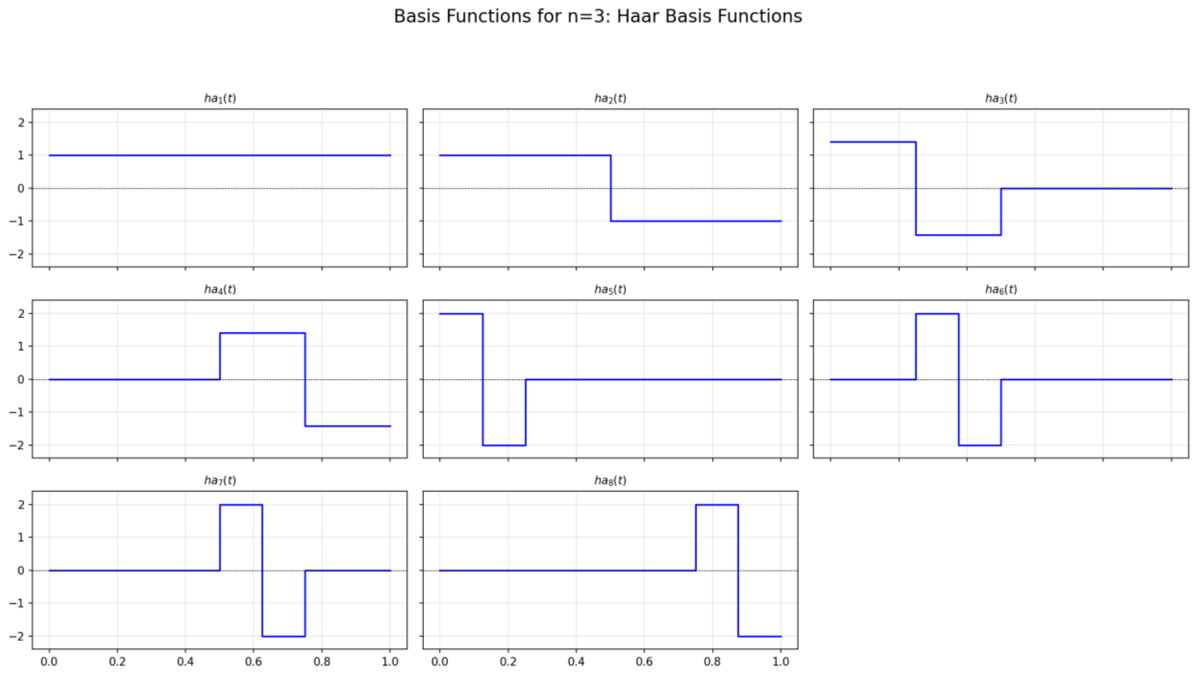


Basis Functions for n=6: Walsh-Hadamard (Sign Change Order)

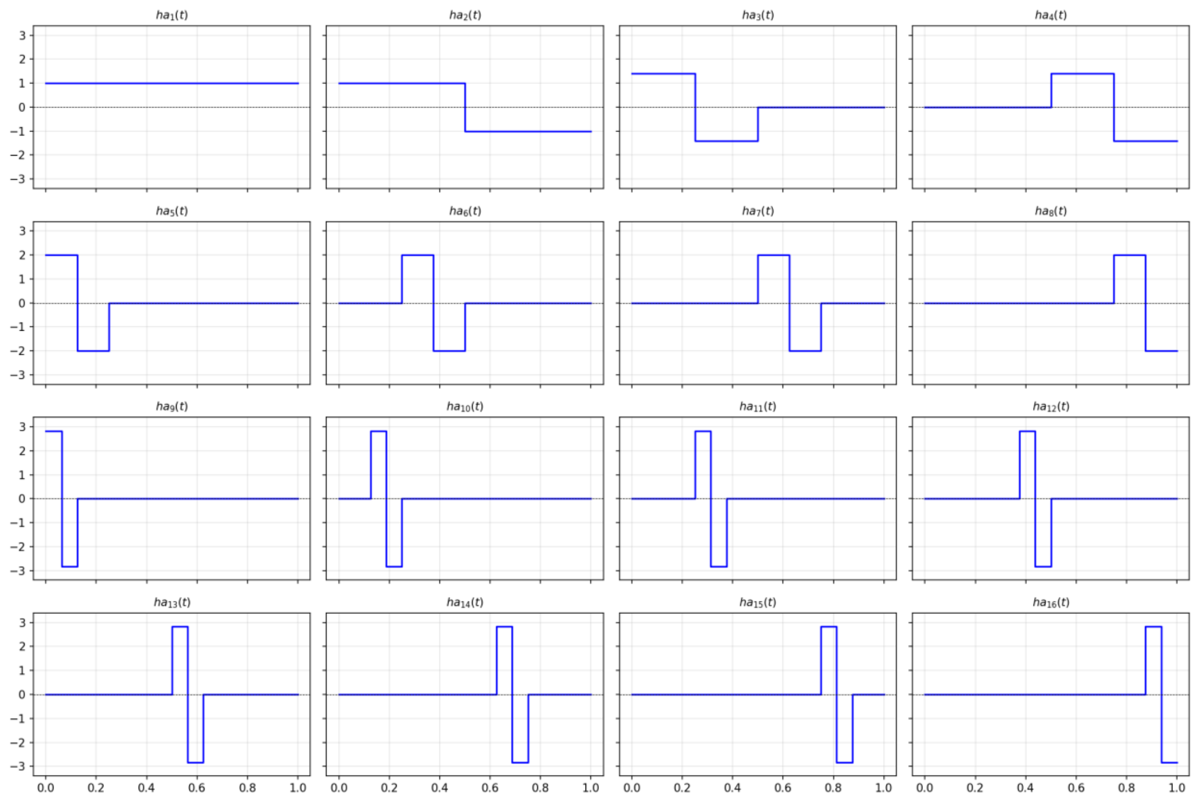


e. We implemented Haar matrix generation as function of n as requested, while preserving the matrix orthonormal. See **haar_matrix** function in **matrices.py** file.
f. We did it by taking the columns of the appropriate Haar matrix multiplied by normalization factor – square of N:

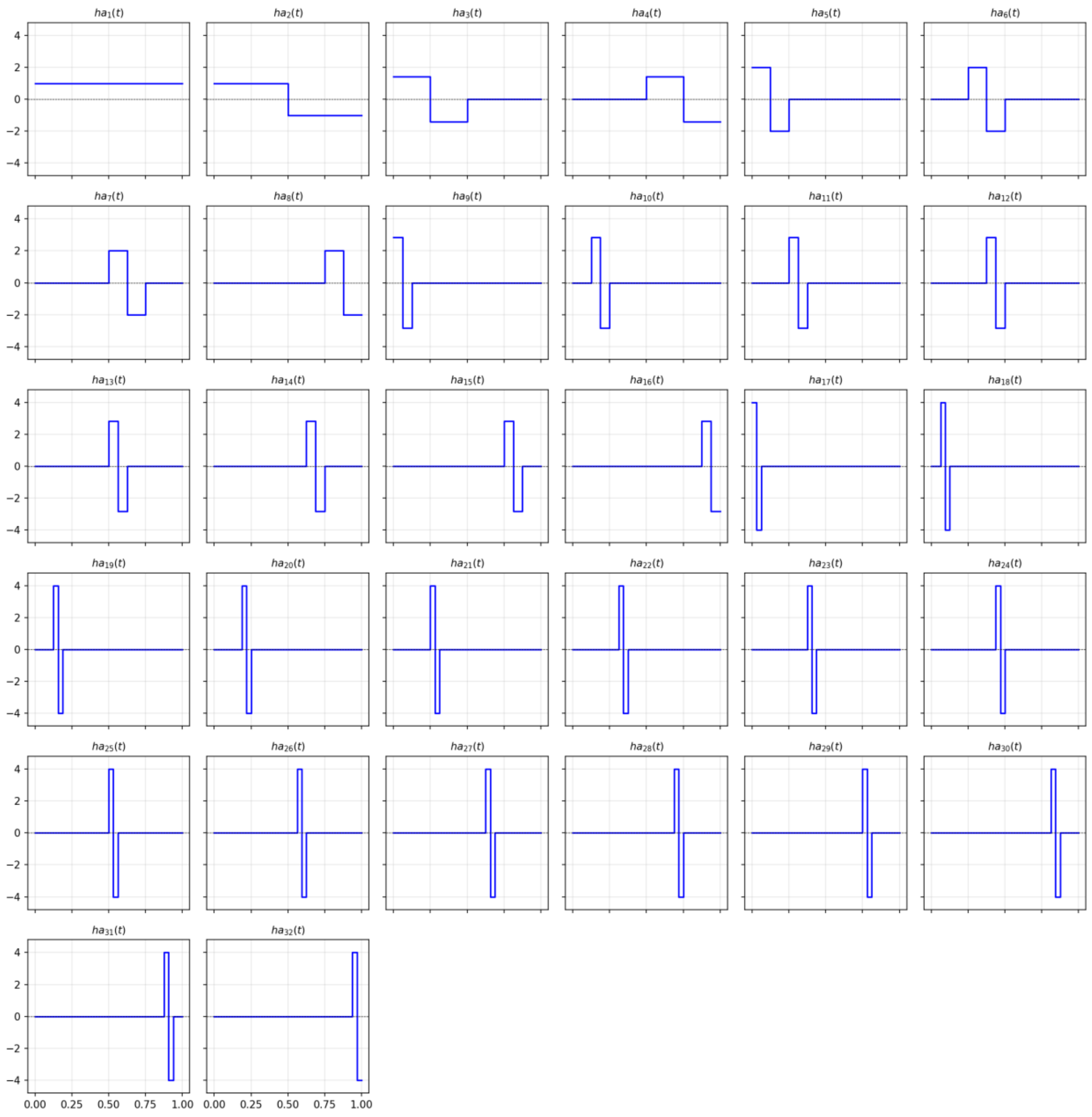




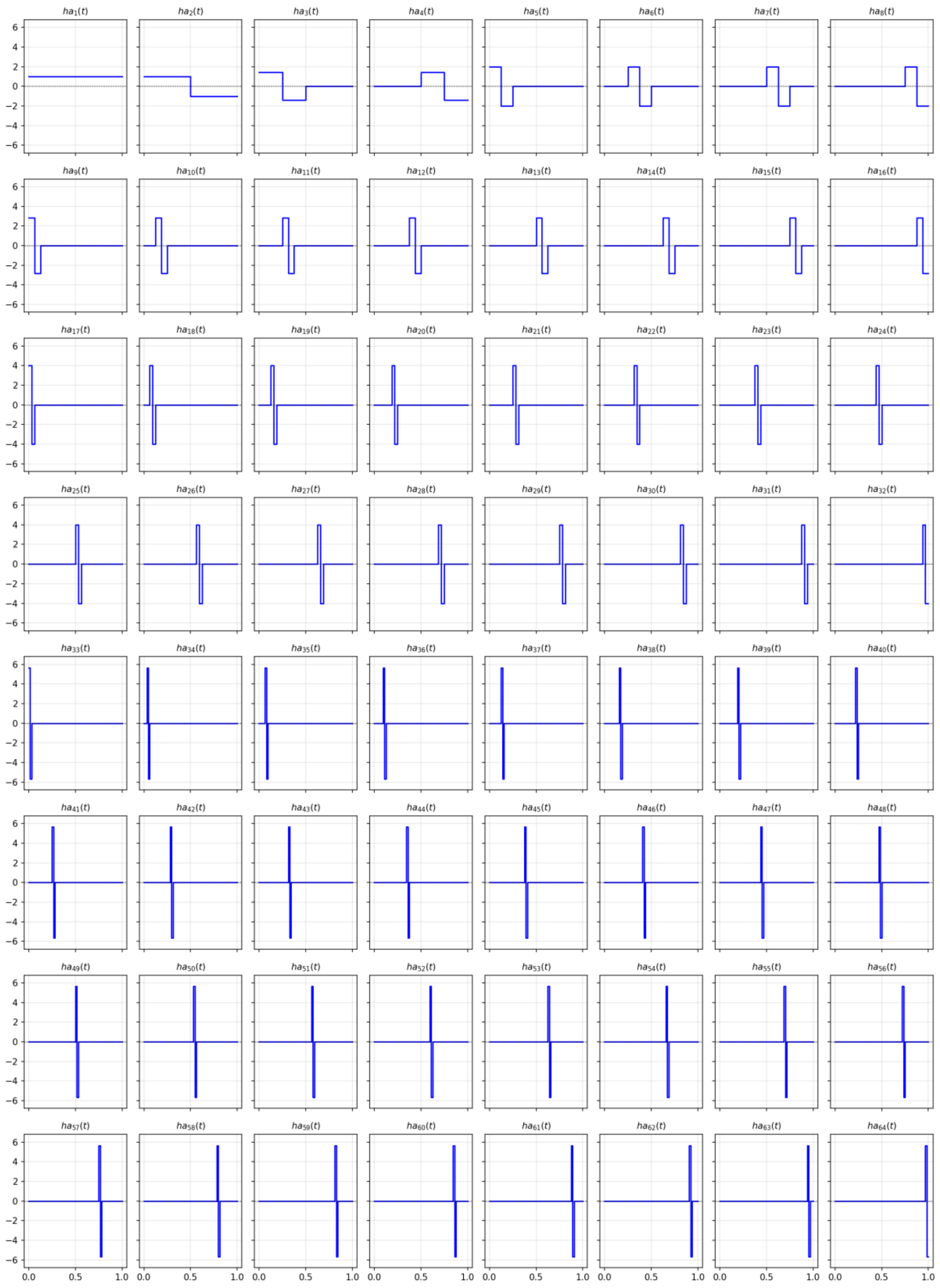
Basis Functions for n=4: Haar Basis Functions



Basis Functions for n=5: Haar Basis Functions

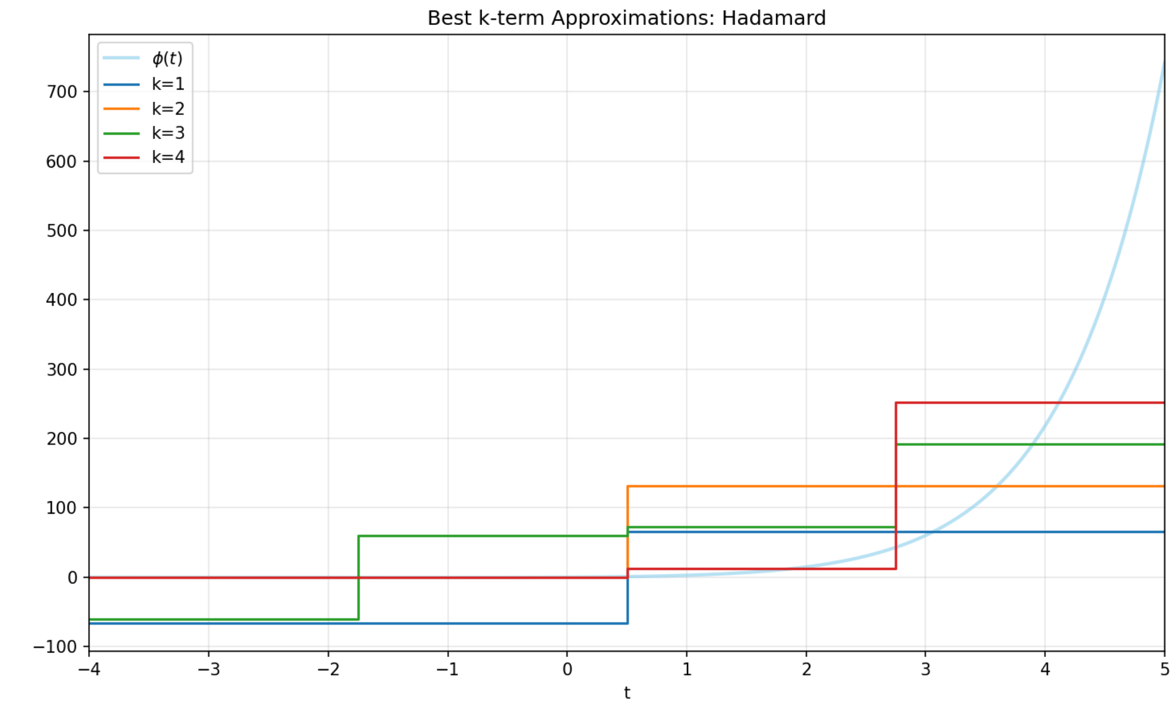


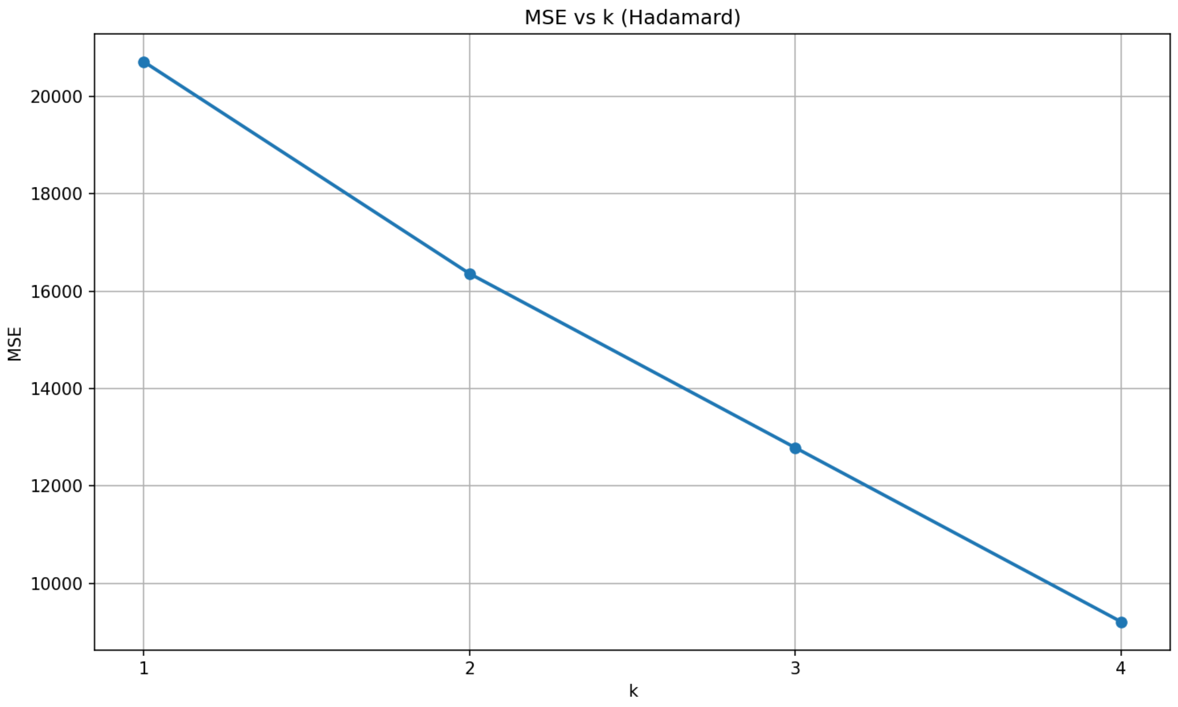
Basis Functions for n=6: Haar Basis Functions



g. We will introduce the results for each one of the bases:

   i. In Hadamard basis:

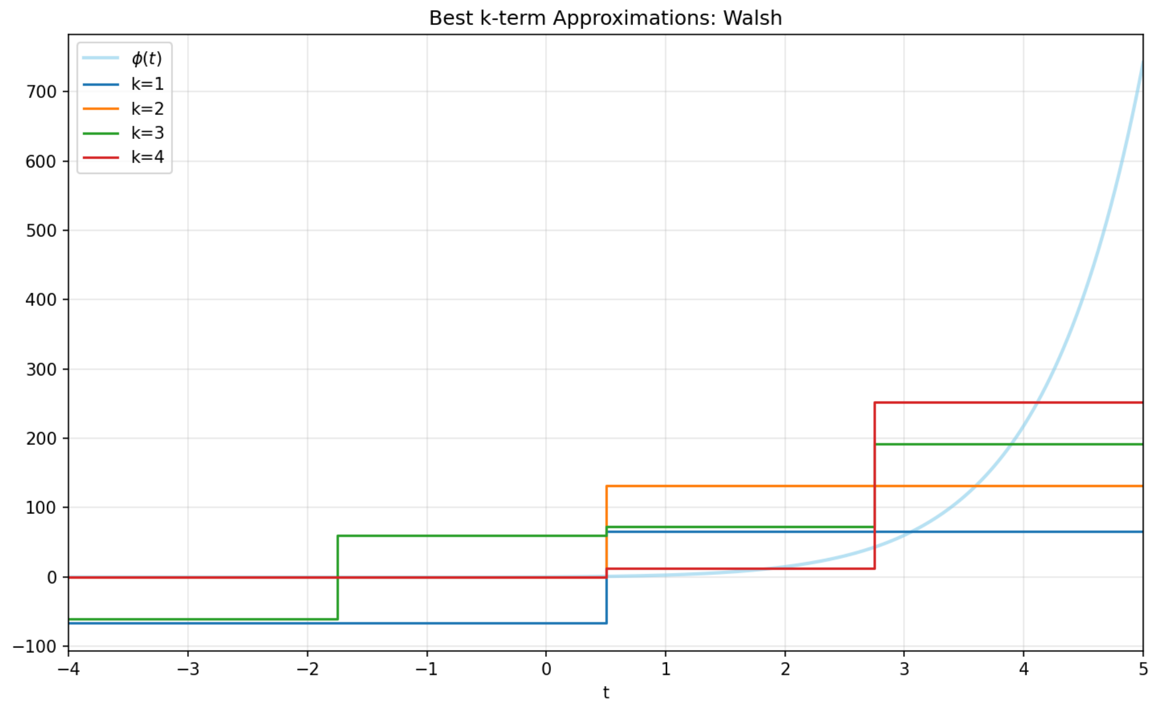


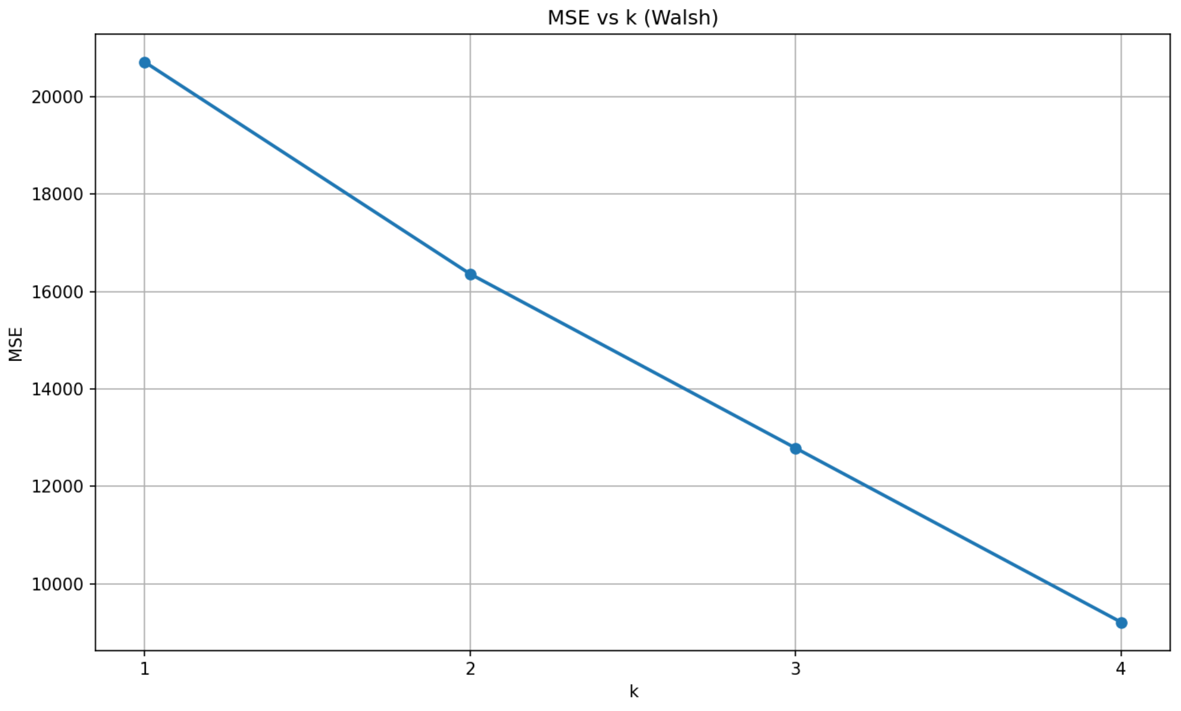


MSE results:

1. 1-term approx.: 20711.93
2. 2-term approx.: 16359.68
3. 3-term approx.: 12784.72
4. 4-term approx.: 9210.81

ii. For Walsh Hadamard basis:

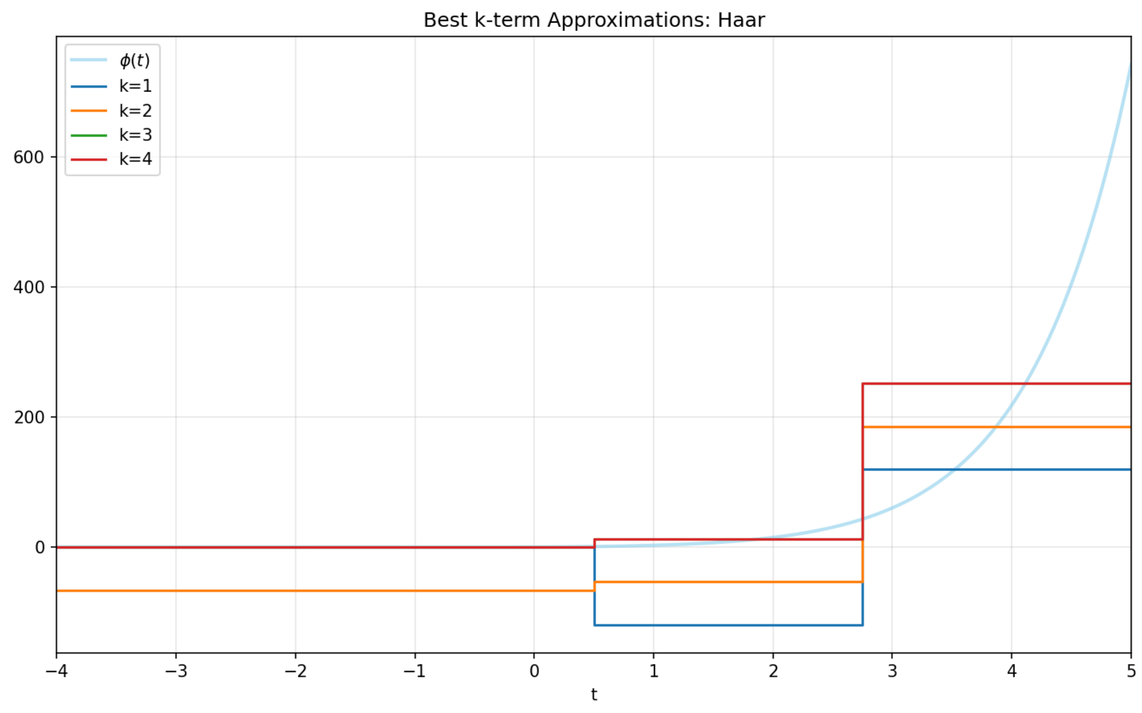


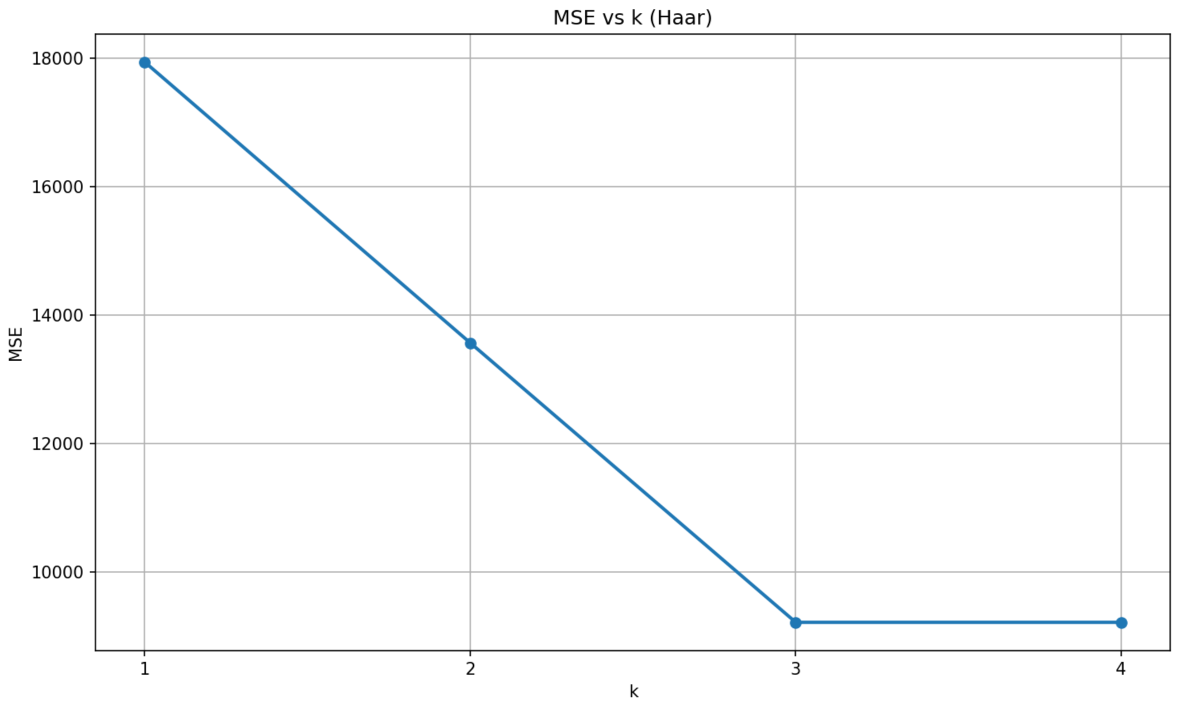


MSE results:

1. 1-term approx.: 20711.93
2. 2-term approx.: 16359.68
3. 3-term approx.: 12784.72
4. 4-term approx.: 9210.81

iii. For Haar basis





MSE results:

1. 1-term approx.: 17936.83
2. 2-term approx.: 13563.06
3. 3-term approx.: 9210.81
4. 4-term approx.: 9210.81

We see that the MSE results for the Hadamard and Walsh-Hadamard bases are identical for all k-term approximations. This is expected, as the Walsh-Hadamard basis is simply a permutation of the Hadamard basis. They consist of the exact same set of vectors, arranged in a different order. Consequently, sorting the coefficients by magnitude selects the same set of basis vectors, resulting in identical reconstruction errors.

In contrast, the Haar basis yields different MSE values for partial approximations (e.g., k=1, 2, 3) because its individual basis vectors differ structurally from the Hadamard family. However, for the full approximation (k=4), all three bases achieve the exact same MSE. This is because all three sets are complete bases that span the exact same vector space. Once all terms are included, the reconstructed signal is equivalent regardless of the basis used.