Question 1

1. The coefficients for different features of house are:

   1.0e+04 * 7.7731 (intercept)
   1.0e+04 * 0.0142 (size of the house)
   1.0e+04 * -0.6362 (number of bedrooms)

   Price of a house with size 1400 square meters and 4 bedrooms = 2.5085e+05

2. Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no 'loop until convergence' like in gradient descent. So, feature normalization doesn't help.

   The coefficients for different features of house (when features are normalized) are:
   1.0e+05 * 3.4922
   1.0e+05 * 4.9330
   1.0e+05 * -0.2545

   The predicted price of a house with size 1400 square meters and 4 bedrooms remains same = 2.5085e+05

3. Mean of size feature = 2.0561e+03
   Mean of number of bedrooms feature = 3.1667
   Mean of prices = 3.4922e+05

   Predicted price for a house with size = Mean of size = 2.0561e+03 and number of bedrooms = Mean of number of bedrooms = 3.1667 is equal to:

   3.4922e+05 = Mean of prices.

   This clearly shows that the mean of features passes through the regression line.

4. Complexity of multiplying two matrices of dimensions A * C and C * B respectively = **O** (A * B * C)
   Now given, A = B = 1000000 and C = 3.
   Multiplying the two matrices would require 1000000 * 1000000 * 3 = 3e12 operations.
   Since this method involves multiplying matrices of these dimensions, the method is clearly not feasible.

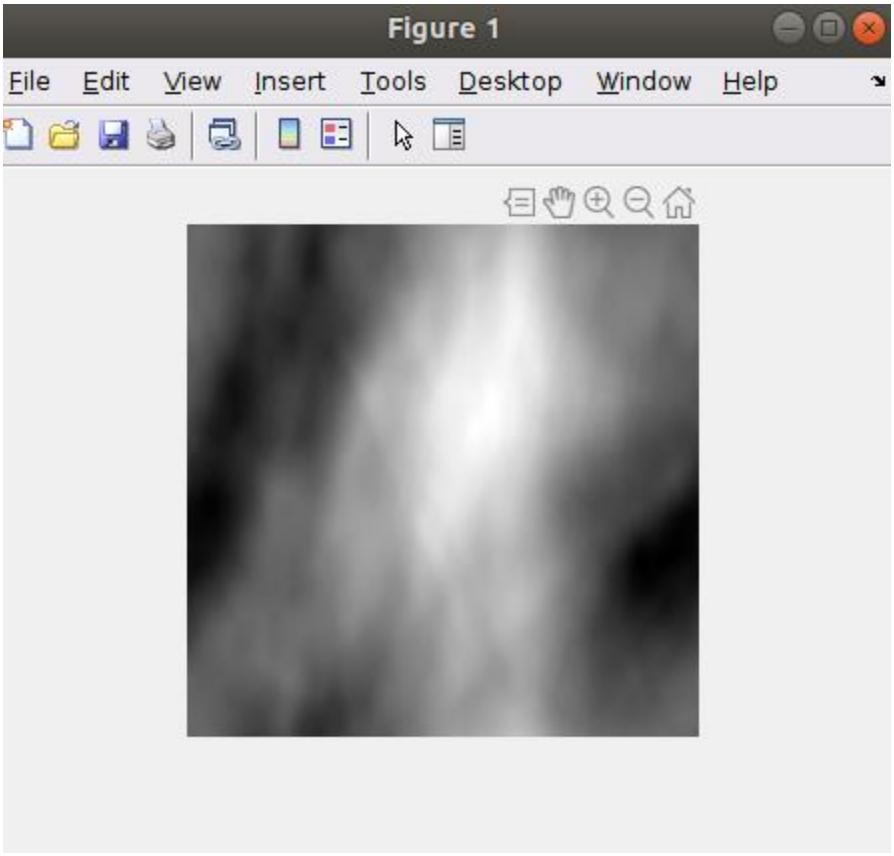
To appreciate how well the model performed we compute the %error given by the formula:

*abs (norm(y_actual, 2) - norm(y_pred, 2)) / norm(y_actual, 2) * 100*

where y_actual is the prices of the houses in test set and y_pred is the predicted prices of the houses in the test set (predicted using the parameters learned in this model)

Question 2

1.



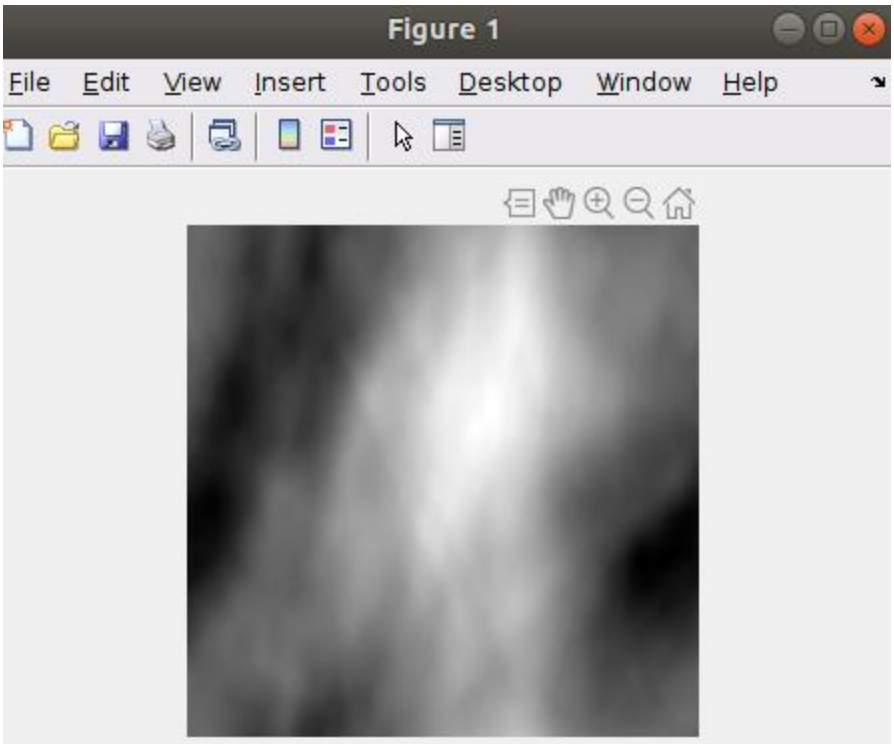


Figure 1
File Edit View Insert Tools Desktop Window Help

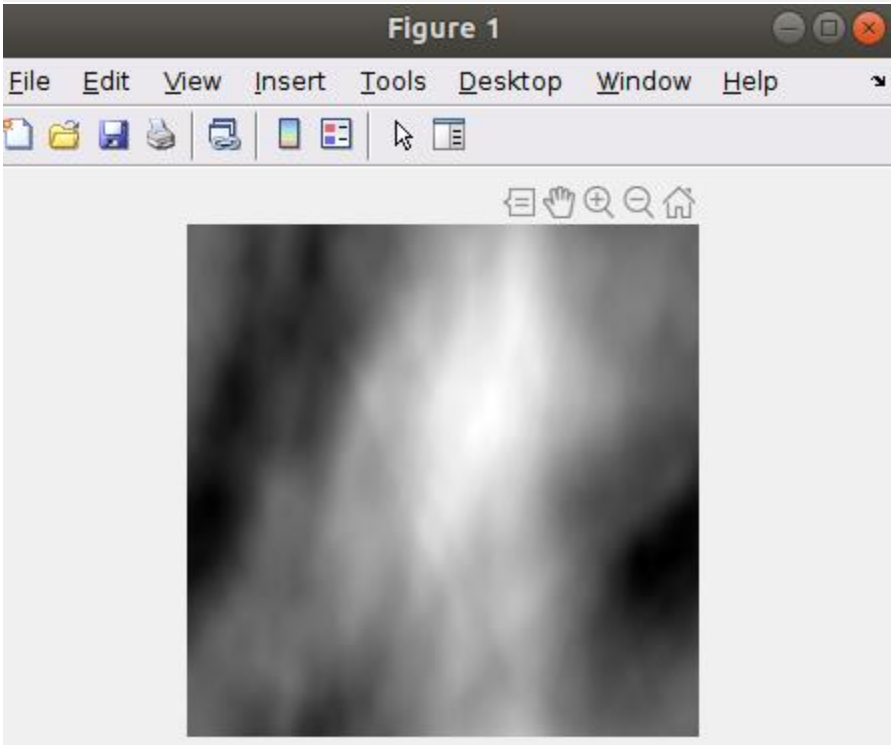
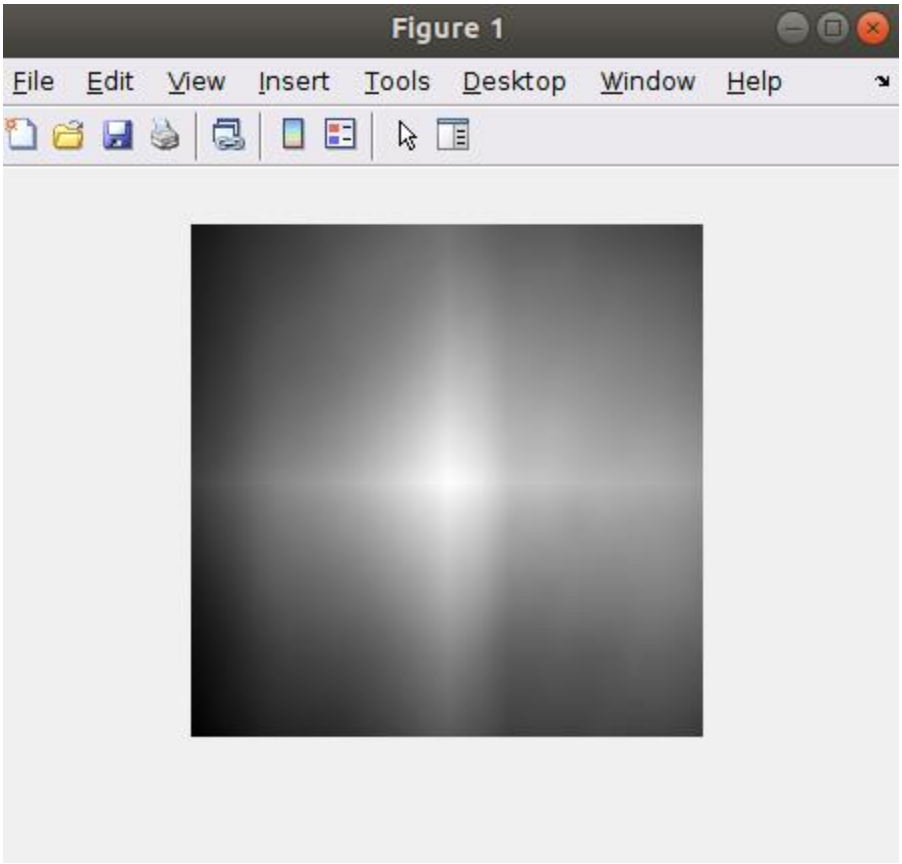
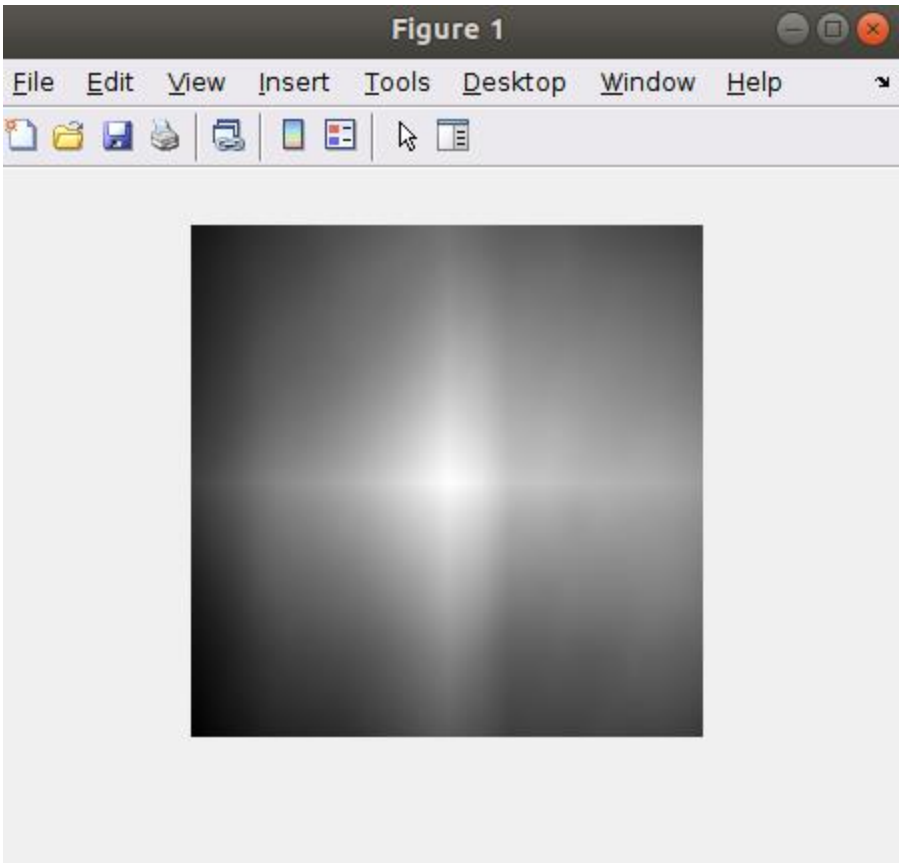


Figure 1
File Edit View Insert Tools Desktop Window Help

Plot of iDFT[FH]





Plot of center portion of f * h(convolution)

Clearly it can be verified that they are not the same.

2. Average of squared difference between pixel values in iDFT[FH] and the central 256 X 256 portion of f * h = 6.6202e+19

3.

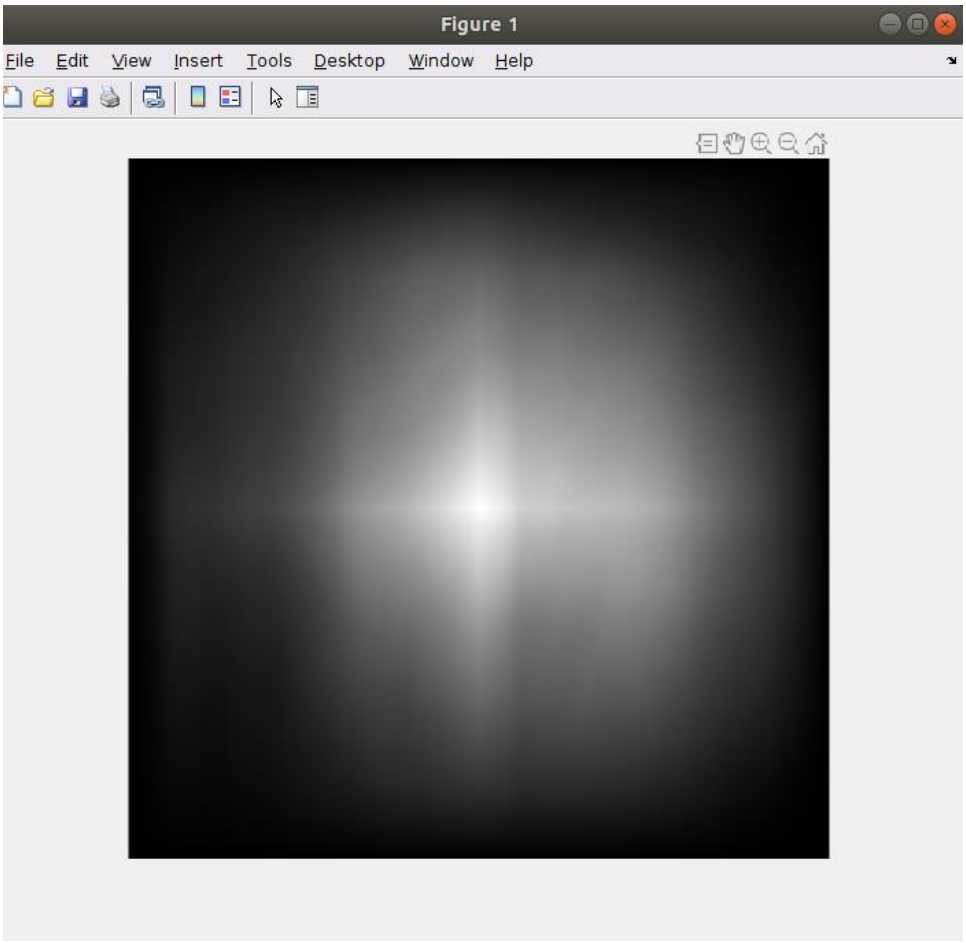


Figure 1
File Edit View Insert Tools Desktop Window Help

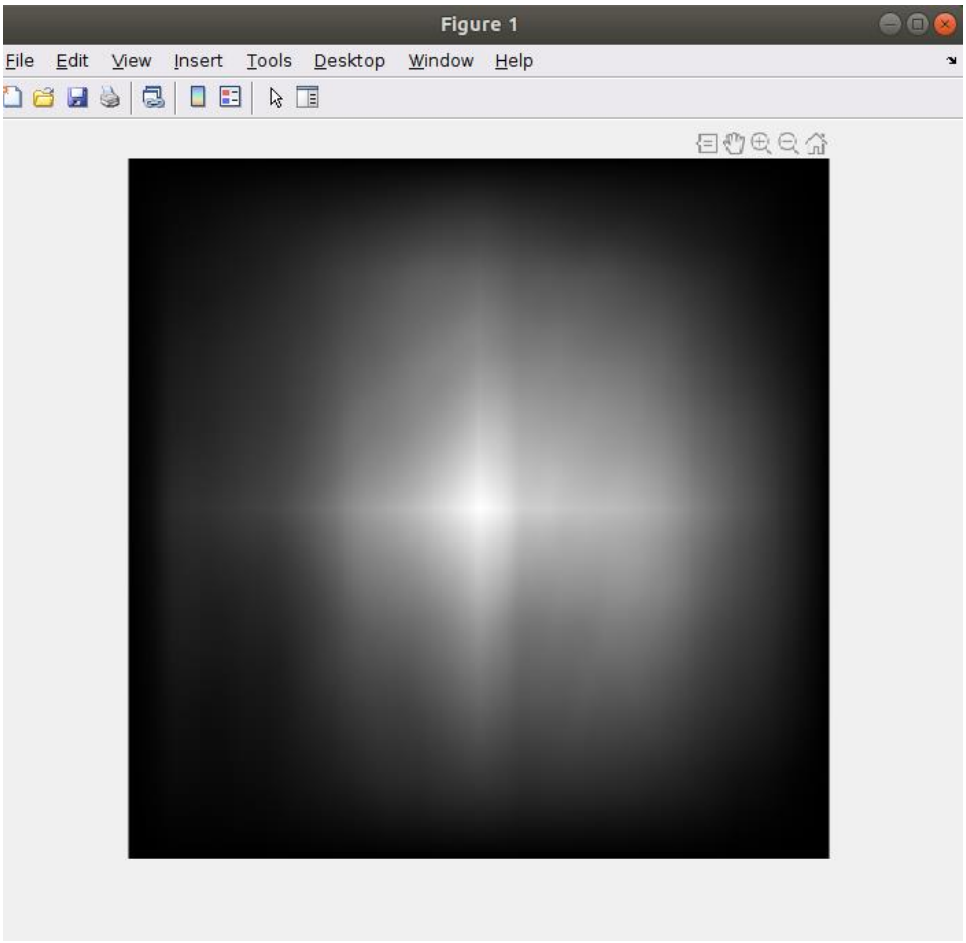
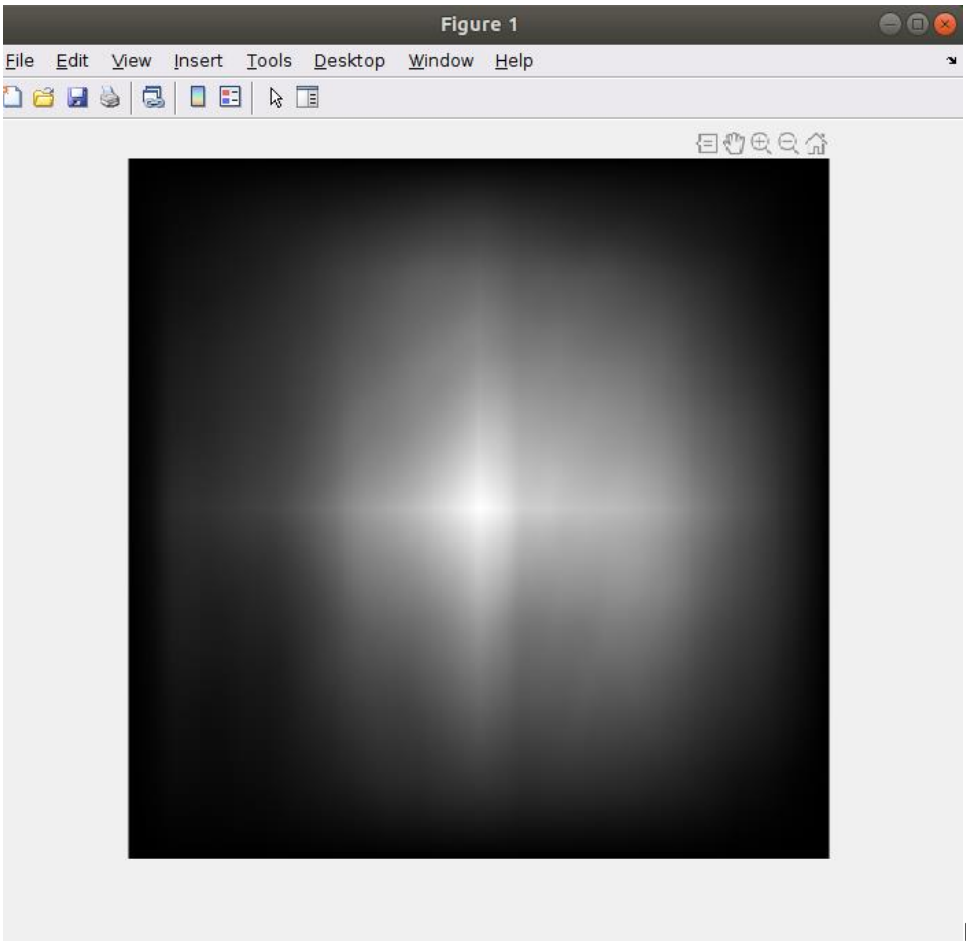


Figure 1
File Edit View Insert Tools Desktop Window Help

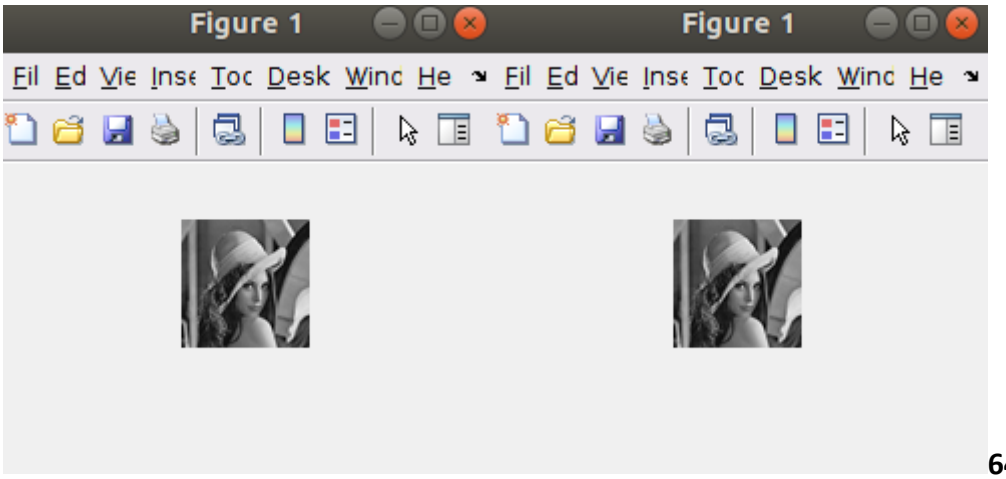


Plot of iDFT[FH] when the images were padded to make them of apt. Dimensions

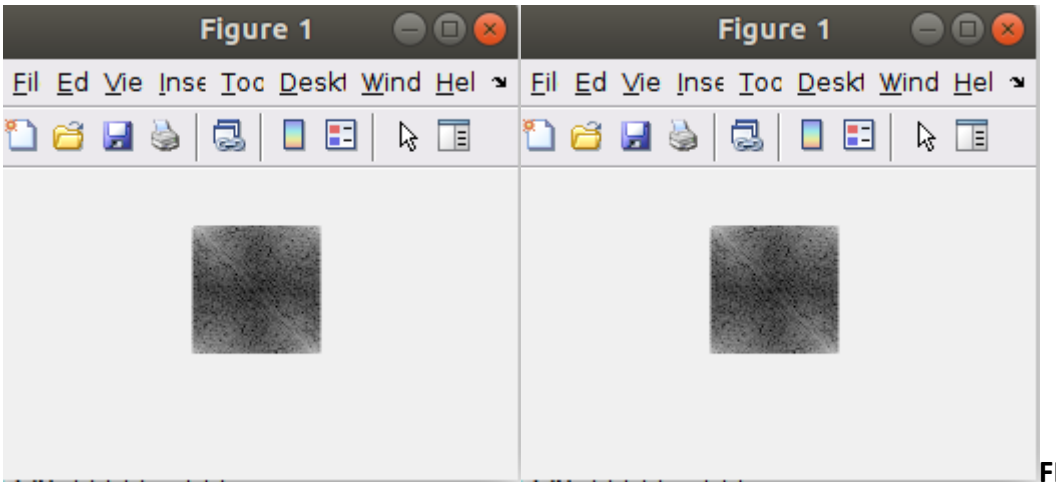
We can clearly see that this plot closely resembles the f * h (convolution) plot.

New average of squared difference between pixel values in iDFT[FH] and the f * h = 2.6085e-12

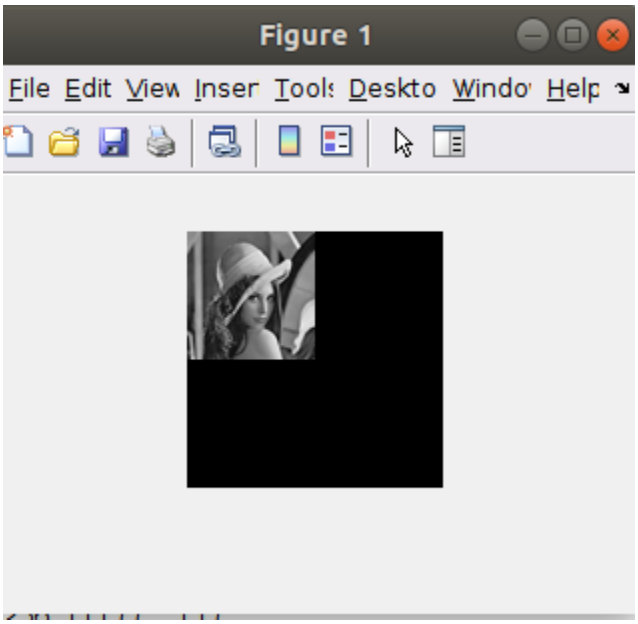
Question 3



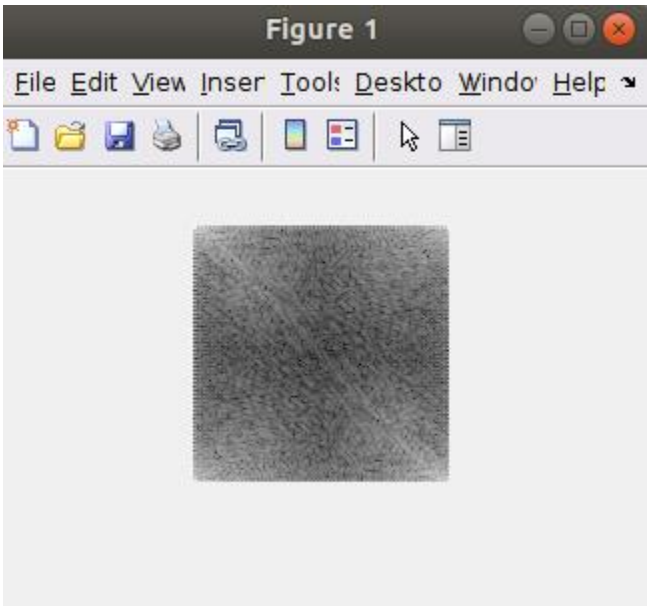
64 X 64 image

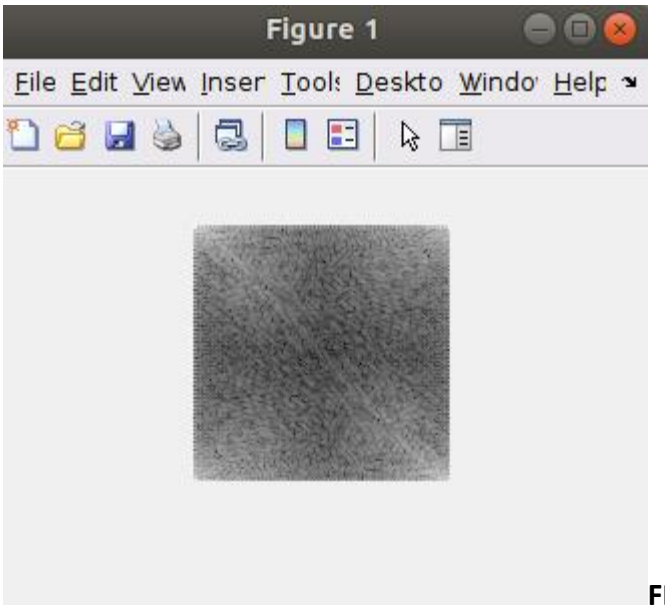


FFT plot of the above 64 X 64 image

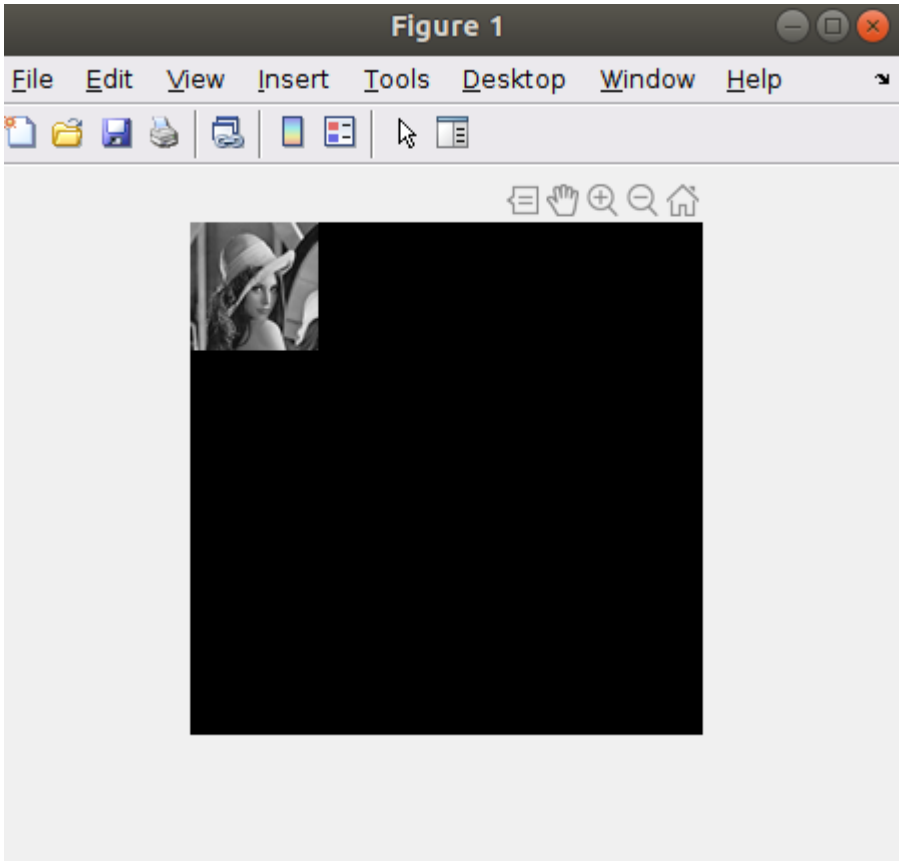


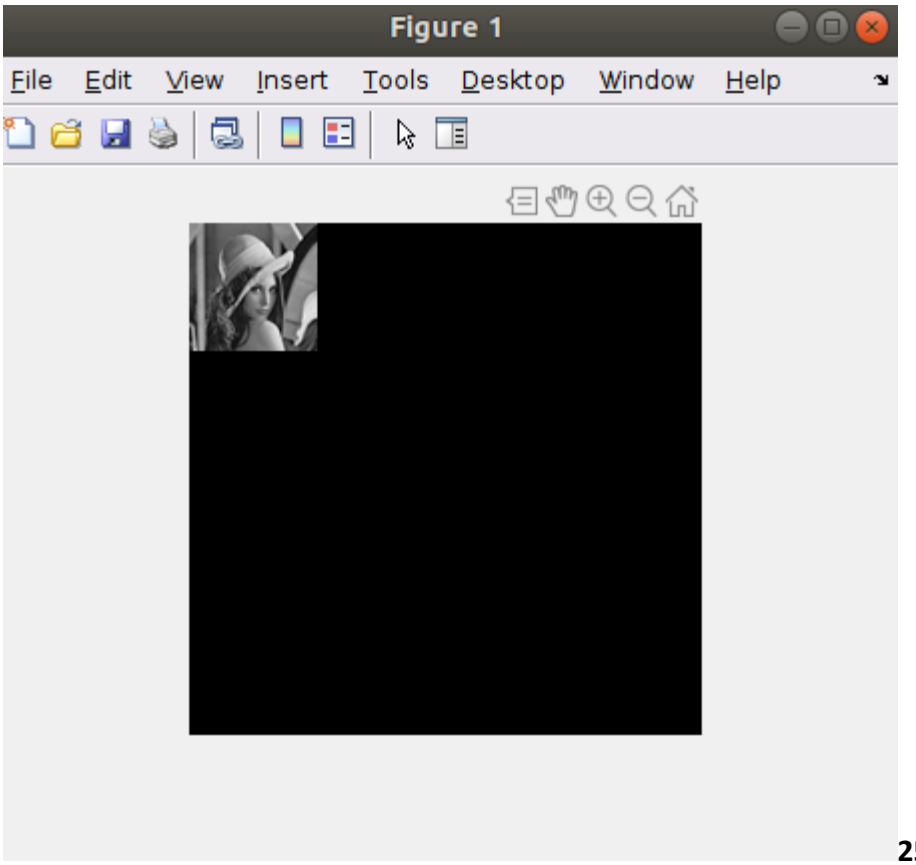
128 X 128 image



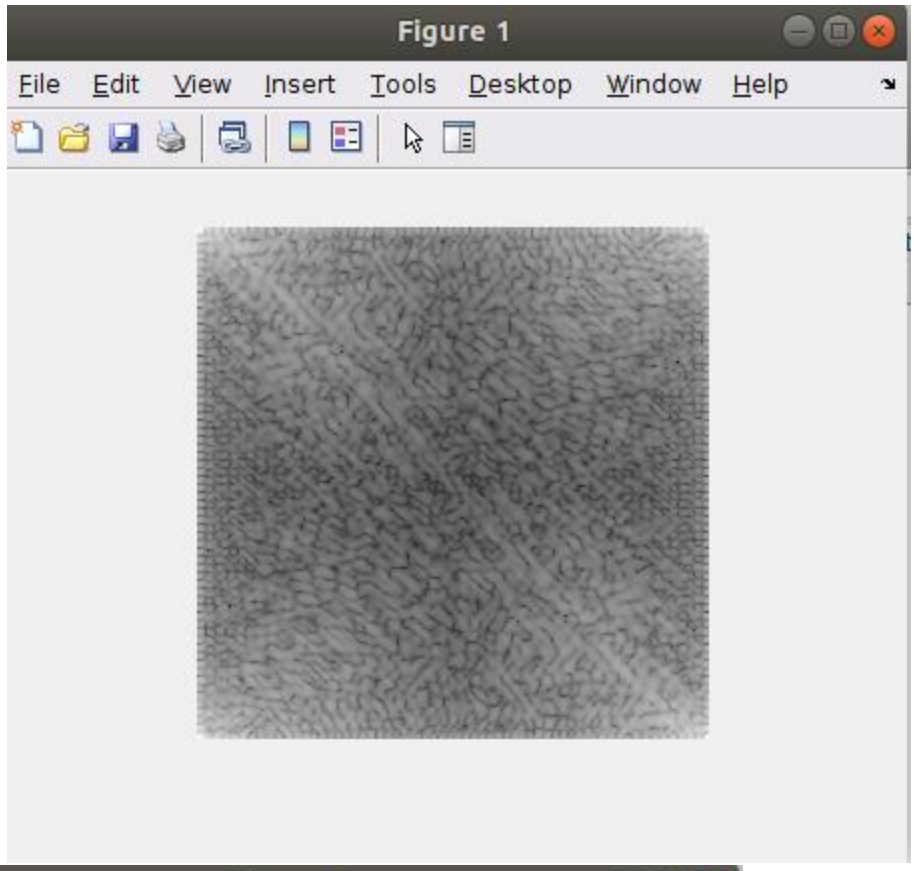


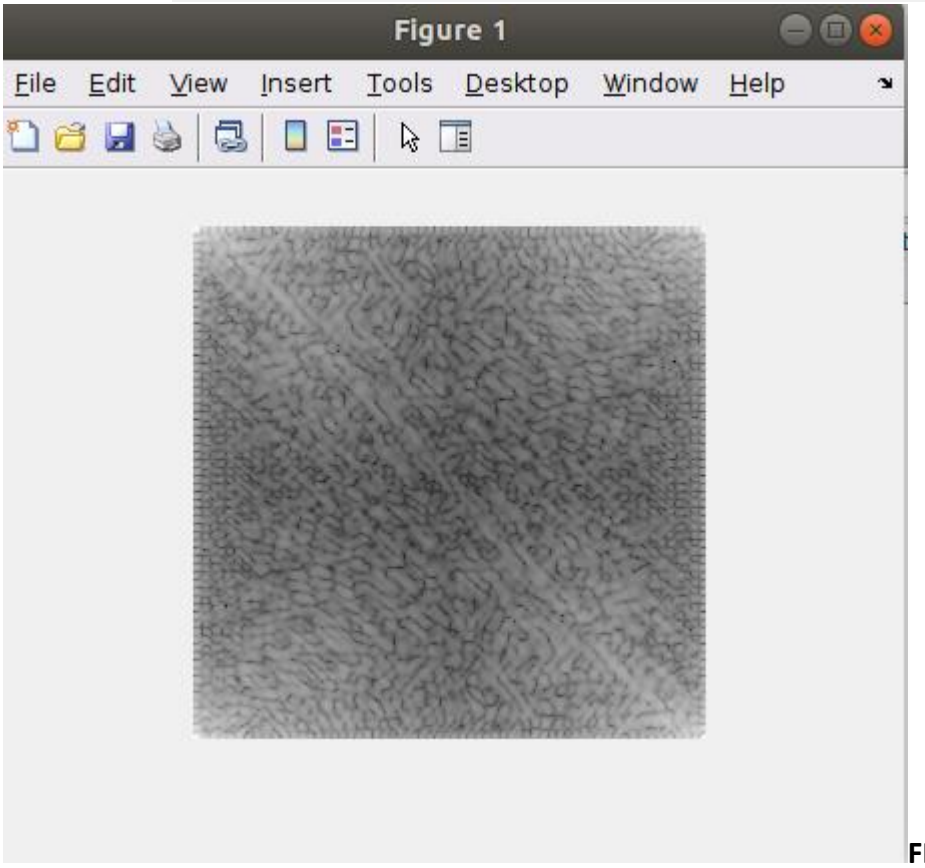
FFT plot of the above 128 X 128 image





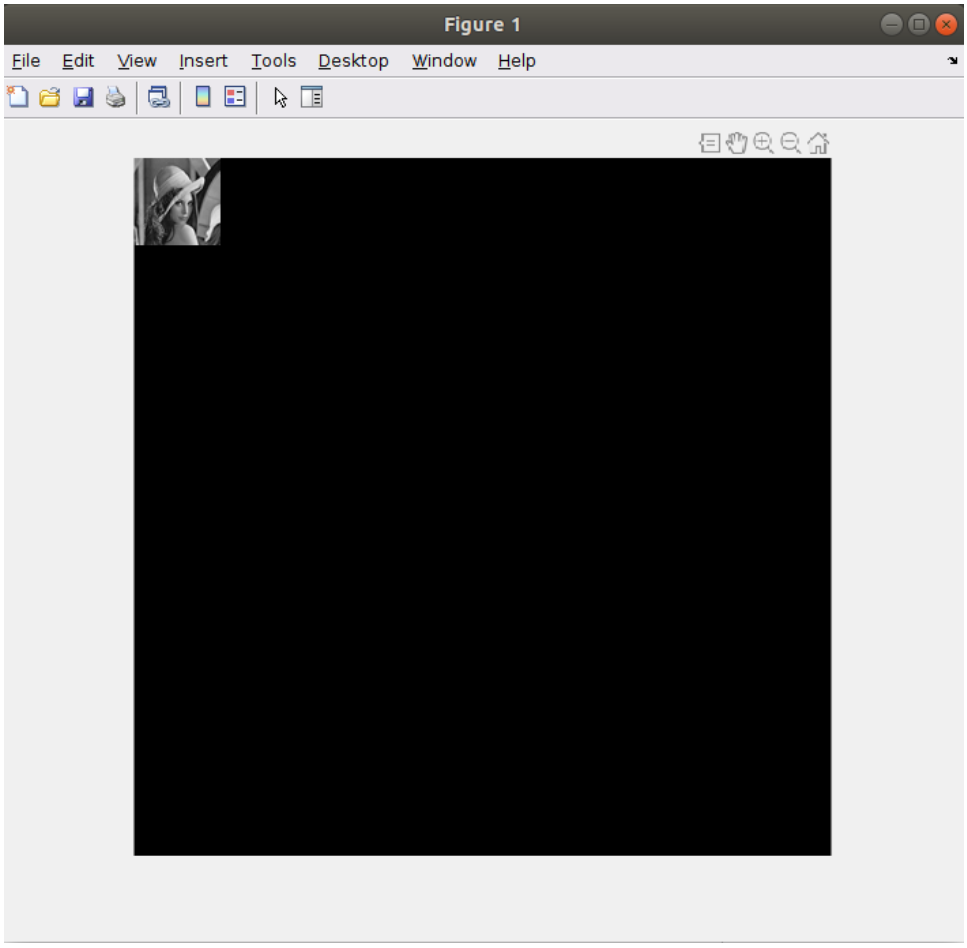
256 X 256 image

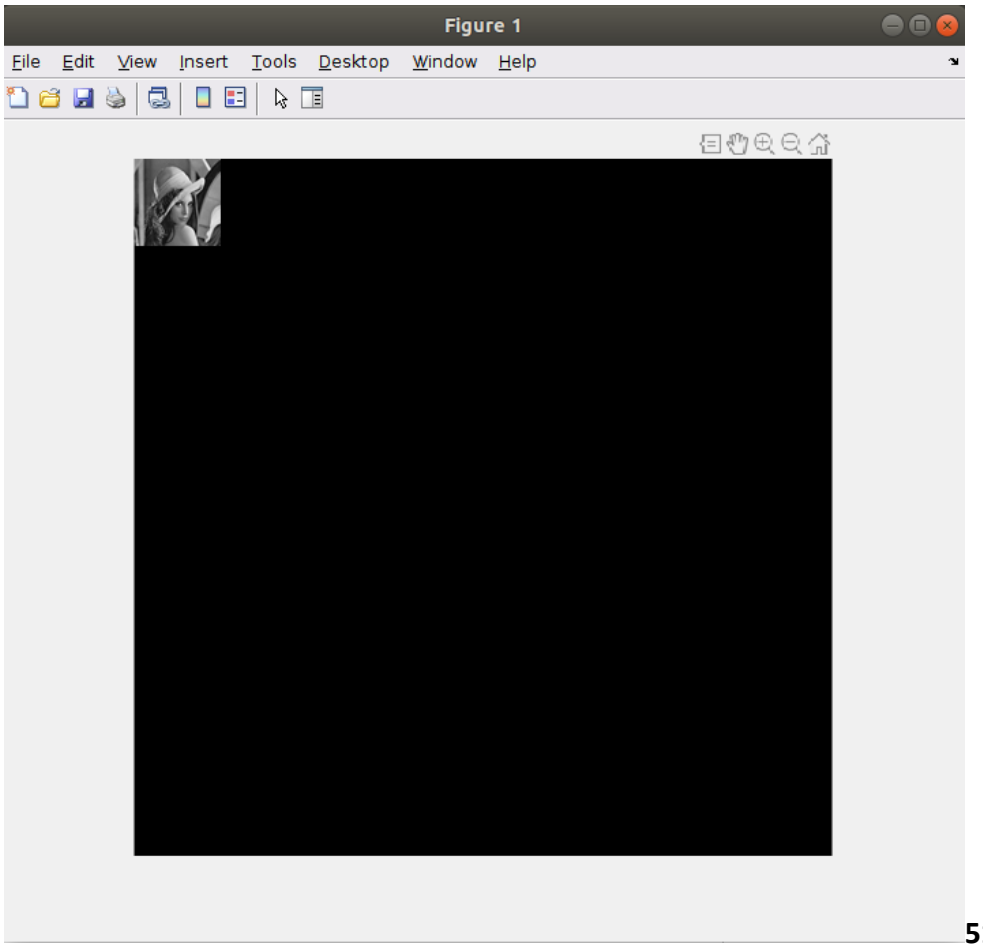




FFT plot of the above 256

X 256 image





512 X 512 image

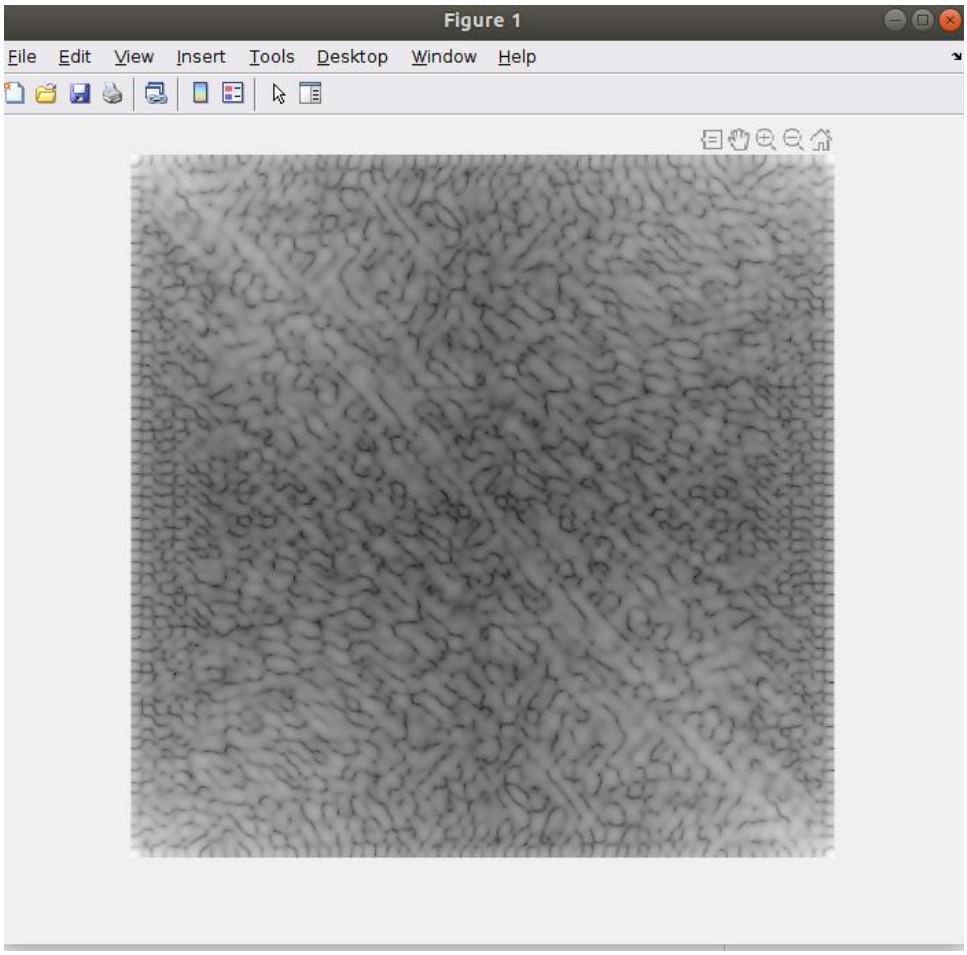
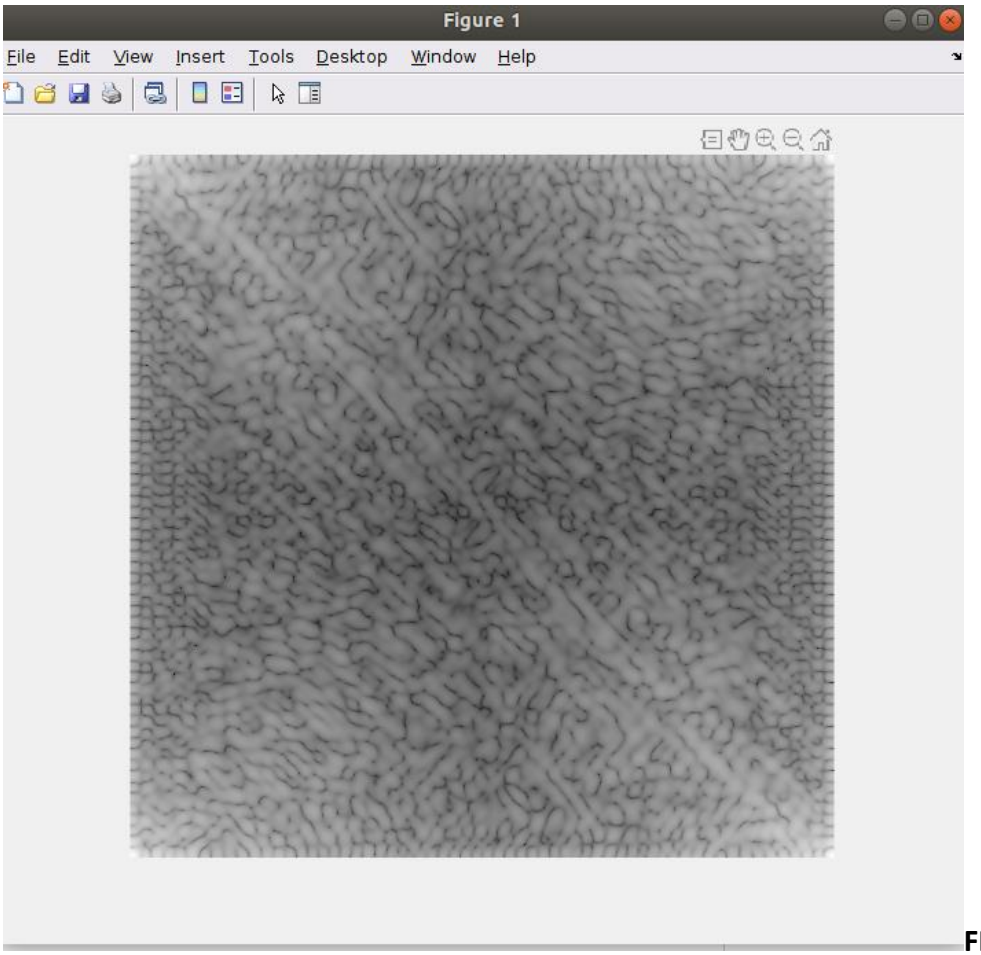


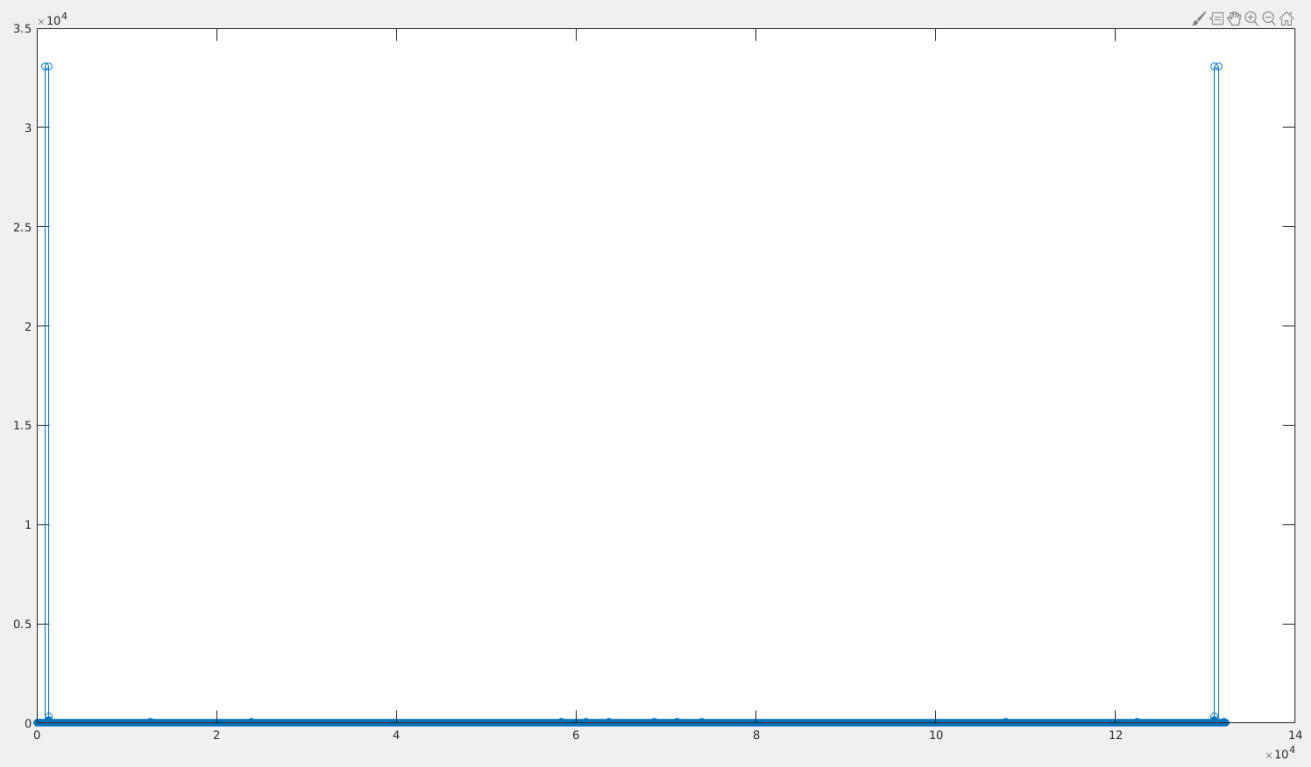
Figure 1
File Edit View Insert Tools Desktop Window Help



FFT plot of the above 512 X 512 image

The FFT plots are similar. The FFT plot corresponding to 512 X 512 image has a higher frequency resolution compared to the 64 X 64 image. This is due to the property of FFT that greater the size greater will be the frequency resolution.

Question 4



FFT plot of the sound signal

The X values read from the plot are 882 and 1321 which corresponds to frequencies of 293 and 440 Hz

Calculations:

882 / 132301 * 44100 = 293Hz

1321 / 132301 * 44100 = 440 Hz

Therefore, the tone consists of two frequencies, 440 Hz and 293Hz.

The noise was cleaned by making all other frequencies apart from these two frequencies (and their corresponding higher frequencies) zero.

Question 5

The correct order of the sequences is: 3, 5, 1, 2, 4.

Taking a cue from the hint, the problem boiled down to checking for each chunks' last 5 seconds which chunk had the greatest similarity in the first 5 seconds (and of course such a chunk cannot be succeeded by any other chunk before). We write a dfs based solution to construct the sequence.

We start with different sequences and find the sequence to be cyclic. As a result, the starting and ending chunks are decided by running a similarity check between each adjacent pair of chunks (from the sequence) and the pair which had the least similarity is the one which corresponds to the starting and ending segment.

Similarity check of two signals is found out using the xcorr function which returns the correlation values of two signals when the second is slide along the first. The maximum value is the point at which maximum similarity occurs.

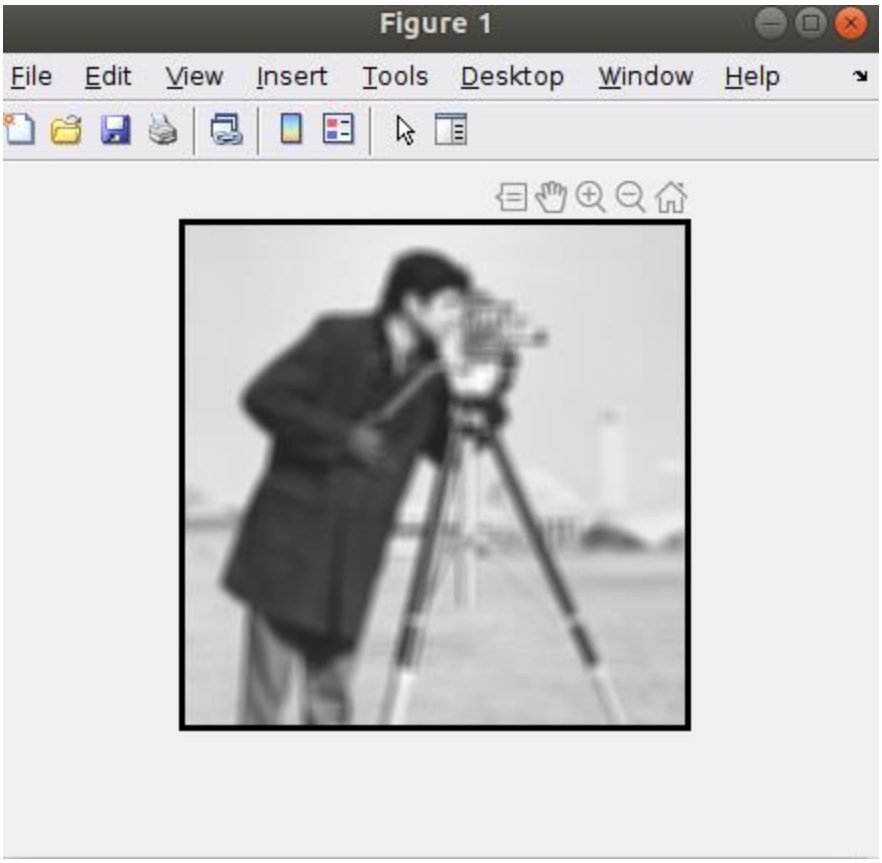
The noise was cleaned by using a median filter of 5th order. Cue to use such a filter was taken from the fact that the noise is salt and pepper and salt and pepper noise was removed using median filter in the previous assignment and as demonstrated in class.

Question 6

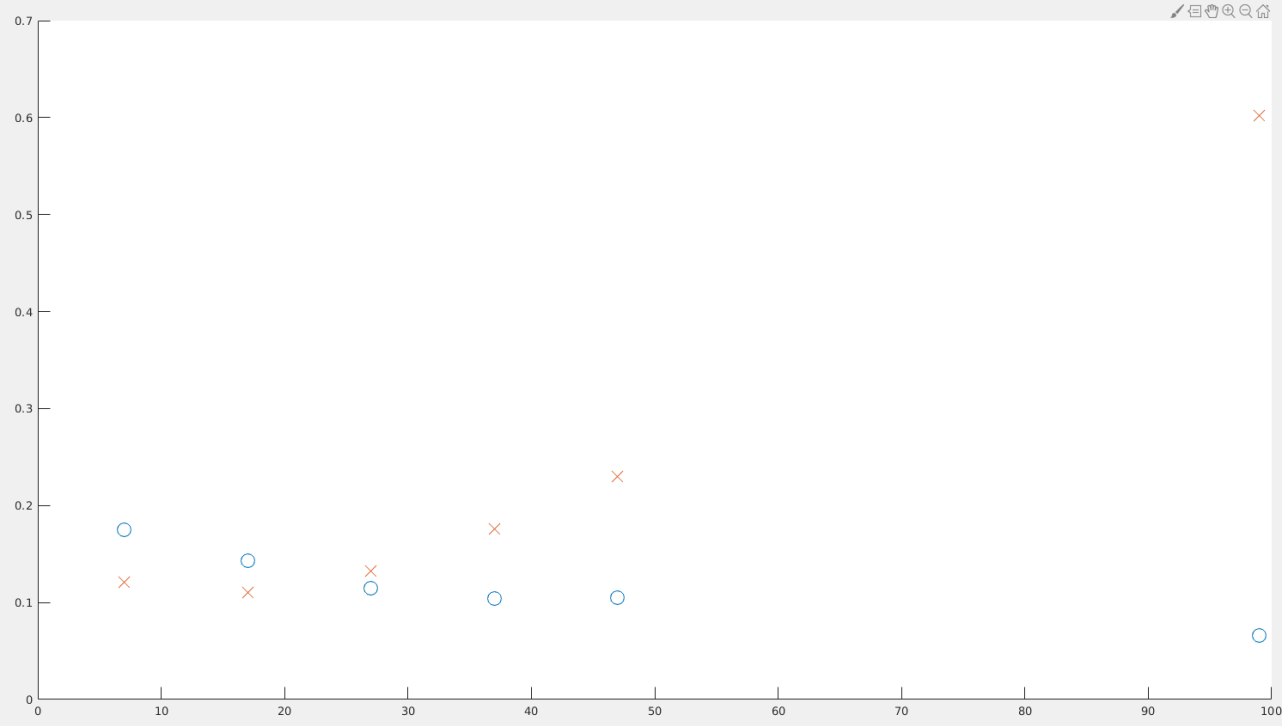
1. The first part involved going through each pixel and then computing it's value by taking average of each of the neighboring pixels' value (pixels within a square of size k with the pixel whose value is being computed at the center of the square)
2. As the filter is moved from one spatial location to the next one, the filter window shares many common pixels in adjacent neighborhoods. In other words, if I use the computed pixel value of the one to the top, I have to just subtract the sum of k pixels and add the sum of k pixels. These k pixels correspond to the newly added row due to shift in spatial location and also the old removed row. This drastically reduces the complexity. Such a technique where we use the previously computed value to compute something new (in order to prevent re computation) is called dynamic programming.

   So, the value for the first pixel is just computed by going through each of the neighboring k*k pixels. The remaining pixels are computed by adding and subtracting rows of k pixels. The first row is an exception where we use the pixel to the left and add/subtract columns of k pixels to compute its value.

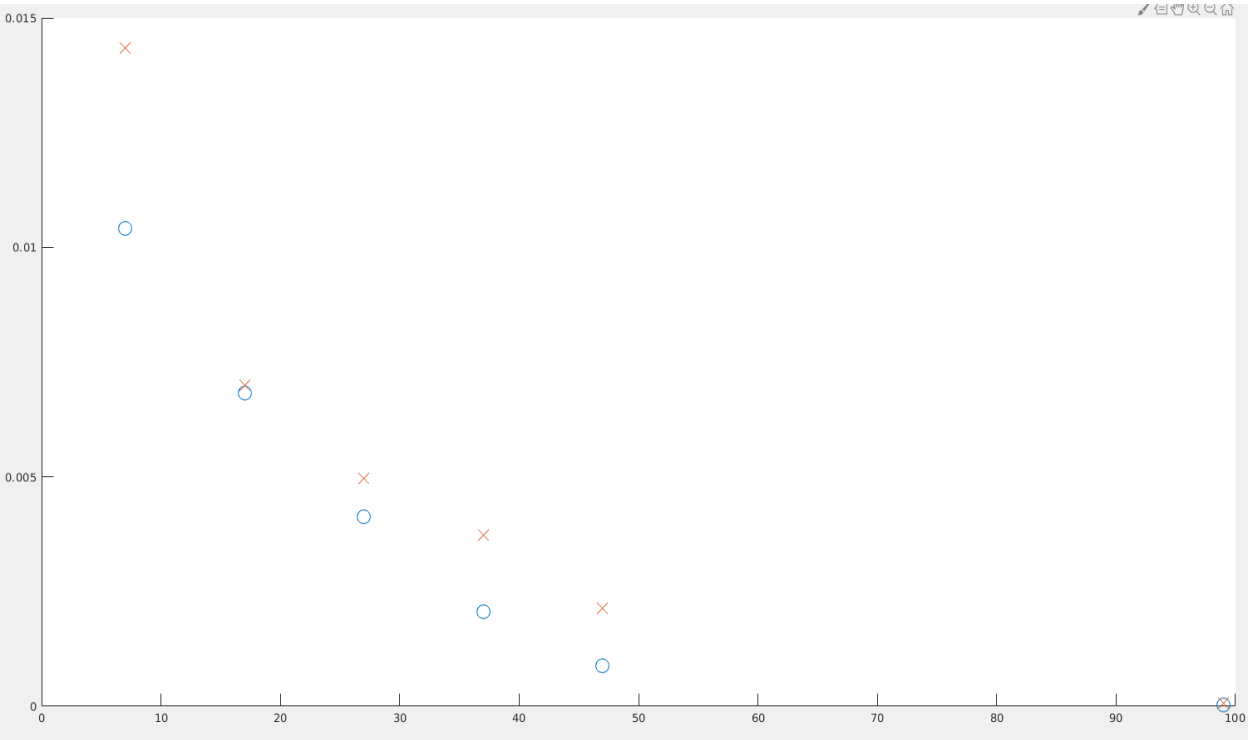
The famous 256 X 256 image of the cameraman



The moving average filter applied on the above image with k = 7



Runtime vs K plot for an image of size 256 X 256. The ‘o’s in blue is for the efficient implementation and the 'x’s in red is for the inefficient implementation



Runtime vs K plot for an image of size 64 X 64. The ‘o’s in blue is for the efficient implementation and the 'x’s in red is for the inefficient implementation

We can clearly see that in almost every case the blue ‘o’s (corresponding to the efficient implementation) perform significantly better than the red ‘x’s (corresponding to the inefficient implementation)

Question 7

We explore the following gradient descent techniques:

Batch Gradient Descent:
Batch gradient descent is a variation of the gradient descent algorithm that calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.

Pros:

1. Fewer updates to the model means this variant of gradient descent is more computationally efficient than stochastic gradient descent.
2. The decreased update frequency results in a more stable error gradient and may result in a more stable convergence on some problems.
3. The separation of the calculation of prediction errors and the model update lends the algorithm to parallel processing based implementations.

Cons:

1. The more stable error gradient may result in premature convergence of the model to a less optimal set of parameters.
2. The updates at the end of the training epoch require the additional complexity of accumulating prediction errors across all training examples.
3. Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
4. Model updates, and in turn training speed, may become very slow for large datasets.

Error rate = 6.3514%

Mini Batch Gradient Descent:
Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

Pros:

1. The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
2. The batched updates provide a computationally more efficient process than stochastic gradient descent.
3. The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

Cons:

1. Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm.
2. Error information must be accumulated across mini-batches of training examples like batch gradient descent.

Error rate = 4.2958%

Stochastic Gradient Descent:

Stochastic gradient descent, often abbreviated SGD, is a variation of the gradient descent algorithm that calculates the error and updates the model for each example in the training dataset.

The update of the model for each training example means that stochastic gradient descent is often called an online machine learning algorithm.

Pros:

1. The frequent updates immediately give an insight into the performance of the model and the rate of improvement.
2. This variant of gradient descent may be the simplest to understand and implement, especially for beginners.
3. The increased model update frequency can result in faster learning on some problems.
4. The noisy update process can allow the model to avoid local minima (e.g. premature convergence).

Cons:

1. Updating the model so frequently is more computationally expensive than other configurations of gradient descent, taking significantly longer to train models on large datasets.
2. The frequent updates can result in a noisy gradient signal, which may cause the model parameters and in turn the model error to jump around (have a higher variance over training epochs).
3. The noisy learning process down the error gradient can also make it hard for the algorithm to settle on an error minimum for the model.

Error rate = 4.6755%