Engineering and Applied Science Programs for Professionals
Whiting School of Engineering
Johns Hopkins University
685.621 Algorithms for Data Science
Mathematical Transformations

This document provides a rollup of the mathematical transformation methods covered in this module. The following list of topics is covered in this module allowing for an understanding of how discrete data is processed to be used in a compact efficient manner for data representation. An introduction to mathematical transformation is provided followed by the definitions of discrete data. Before going into the transformation, discrete data is described with an example of one dimensional data and a second example of two dimensional data. Once the discrete data is level set, the descrete mathematical transformation are described in detail.

Contents

1 Introduction to Mathematical Transformations

Data is everywhere. Nowadays, it is most likely in digital form that can be processed by a machine. In mathematics, certain transforms are particularly useful because they come with well-defined inverse transforms, allowing you to switch back and forth between different representations of a function or signal. The mathematical transformation cycle is a series of operations on data, mainly including data collection, data input, data transformation, and the data output into a new space. Data transformation may include but is not limited to retrieval, transformation, and/or classification in hopes to produce meaningful information in the new space. The term "mathematical discrete transformations" can refer to the mathematical properties or structures that are inherent to the transformation itself. It could mean transformations that only deal with discrete values by their mathematical nature. For example, matrix transformations on finite fields, the transformation itself is inherently "discrete" because it only deals with integers. Another example is the permutation functions, which only take on discrete values and are thus inherently "mathematical discrete transformations". The term "discrete mathematical transformations" are a subset of mathematical transforms usually refers to transformations applied to discrete data or signals. These are particularly useful in digital signal processing, image analysis, and data science. In this context, "discrete" describes the nature of the data or the domain over which the transformation is applied. For example, Fourier Transform vs. Discrete Fourier Transform (DFT), while the Fourier Transform is for continuous signals, DFT is applied to discrete signals. Another example, the Discrete Cosine Transform (DCT), used in image compression algorithms like JPEG. The ability to perform inverse transformations allows for perfect reconstruction of the original data from the transformed data, which is useful in applications like data compression, filtering, and more. Figure 1 shows a discrete mathematical transformation using the Discrete Wavelet Transform.

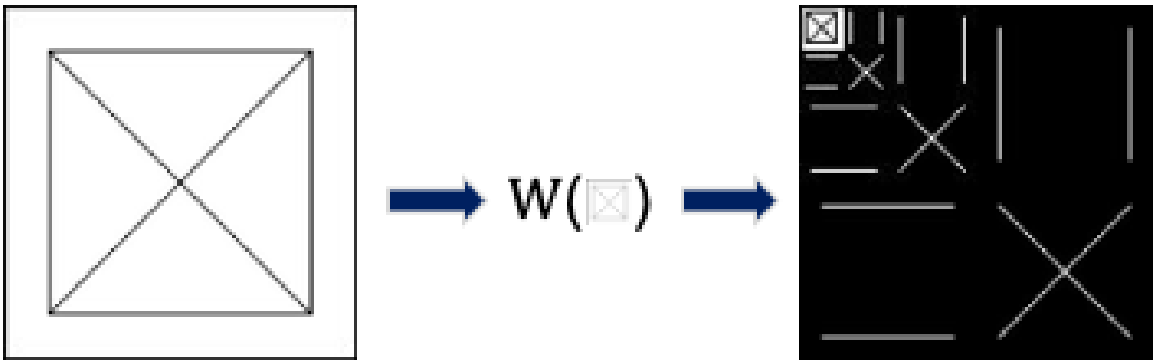


Figure 1: 2-Level wavelet example with an input image containing vertical, diagonal and horizontal edges passed through the wavelet transformed showing the output transform.

In this example of the discrete mathematical transformation the input image in Figure 1 is used to show a visual representation of a 2-level wavelet. There are several forms of discrete mathematical transformations which include the mapping of the data from an input space to a new space. While there are several goals for discrete mathematical transformations in the study of data science, specifically in this course, the goal is to represent a larger data set by a small number of values representing the data. This is shown in Figure 2

The wavelet example is meant to show how a transformation can be used to show the image in a much smaller format. Careful analysis of the transform shows that the mapping also allows extraction of the vertical, diagonal and horizontal edges of the input image.

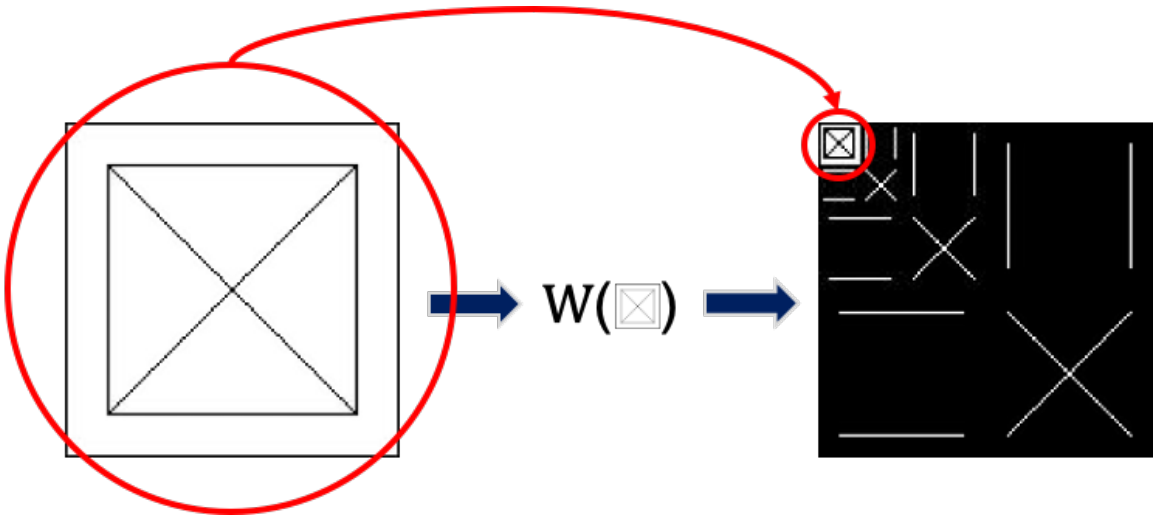


Figure 2: 2-Level wavelet example mapping from a larger input space to a small space allowing for the image to be represented by $1/16^{th}$ the size.

2 Defining Discrete

In the context of AI, Data Science, and Data Analytics, the term "discrete" often refers to data or variables that can take on only a finite or countable number of values. This is in contrast to "continuous" data or variables, which can take on any value within a defined range. When talking about "discrete mathematical transformations," the term "discrete" generally refers to the nature of the data, the domain, or the transformation itself.

2.1 Discrete in AI

In the context of Artificial Intelligence (AI), the term "discrete" typically refers to data, variables, or spaces that can take on only a finite or countable set of distinct values. This is in contrast to "continuous" elements, which can assume any value within a certain range. Understanding the concept of "discreteness" is crucial in AI for model selection, data representation, and algorithm design.

Discrete in AI Algorithms and Models

Decision Trees and Rule-Based Systems: These often work with discrete attributes or features, such as "Yes" or "No", "High/Medium/Low", etc.

Search Algorithms: Algorithms like A* or Dijkstra's algorithm often operate in a discrete search space, where the paths, nodes, or states are distinct.

Natural Language Processing (NLP): Text data is inherently discrete, comprising a finite set of letters, words, or other symbols.

Classification Problems: The output labels are usually discrete (e.g., spam/not spam).

Reinforcement Learning: In some cases, the action space is discrete, such as moving a chess piece to a specific position on the board.

Discrete Data Types

Categorical Data: Non-numeric data like names, labels, or categories.

Ordinal Data: Discrete data that have an inherent order, such as rankings or ratings, but the intervals between the ranks may not be uniform.

Boolean Data: Takes on one of two discrete values, often represented as True/False or 1/0.

Count Data: Represent quantities that can only take on integer values (e.g., the number of occurrences).

2.2 Discrete in Data Science

In Data Science, the term "discrete" refers to data or variables that can take on a finite or countable set of distinct values. This is in contrast to "continuous" data or variables, which can take on an infinite number of values within a given range. The concept of "discrete" plays a significant role in many aspects of data science, from data collection and preprocessing to modeling and analysis.

Discrete Data Types

Categorical Data: These are non-numeric data that fall into specific categories, such as gender, car brands, or types of fruit.

Ordinal Data: These are discrete but have an inherent order. For example, star ratings for a product (1, 2, 3, 4, 5) are ordinal.

Boolean Data: Takes on one of two discrete values, typically represented as True/False or 1/0.

Count Data: Represents quantities that can only take on natural number values, like the number of customers visiting a store.

Applications in Data Science

Classification Problems: Target variables are often discrete (e.g., churn/no churn, fraud/no fraud).

Time Series Analysis: While time can be continuous, it is often discretized into intervals for easier analysis.

Text Mining: Words or characters are discrete elements in text data.

Customer Segmentation: Often involves clustering based on discrete attributes like purchased/not purchased, clicked/not clicked, etc.

Decision Trees and Random Forests: These algorithms are particularly well-suited for handling discrete variables.

Discrete Mathematical Transformations

One-Hot Encoding: Often used to transform categorical data into a format that could be provided to machine learning algorithms.

Integer Encoding: Another method for encoding categorical variables, where each unique category is mapped to an integer.

Bin Counting: A technique where continuous variables are discretized into 'bins' to simplify analysis or to prepare data for certain algorithms.

2.3 Discrete in Data Analytics

In the context of Data Analytics, the term "discrete" refers to data or variables that can assume a finite or countable number of distinct values. Unlike "continuous" data, which can take on an infinite number of values within a given range, discrete data are characterized by clear separations between each possible value.

Discrete Data Types

Categorical Data: These are variables like "gender," "marital status," or "customer segment," which can take on one of a limited, fixed set of options.

Ordinal Data: These are discrete data types with an inherent order, such as "low," "medium," and "high" ratings, although the intervals between the values are not uniform.

Boolean or Binary Data: This is data that can take on one of two values, such as "yes" or "no," or "1" or "0."

Count Data: These are variables that represent the number of occurrences of an event, such as the number of clicks on a website or the number of sales transactions.

Applications in Data Analytics

Descriptive Statistics: Descriptive metrics like mode or frequency distributions are often used for analyzing discrete variables.

Data Visualization: Discrete data are often visualized using bar charts, pie charts, or histograms with distinct categories.

Clustering and Segmentation: Discrete variables are commonly used for segmenting customer bases or clustering similar data points.

A/B Testing: In this form of hypothesis testing, the outcome is often a discrete variable (e.g., click/no-click, convert/don't convert).

Pattern Recognition: Identifying regularities or anomalies in a dataset often involves examining discrete variables like event codes or status flags.

Discrete Mathematical Transformations

One-Hot Encoding: This is a method used to convert categorical data variables so they can be provided to machine learning algorithms as inputs.

Binning: Continuous data can be transformed into discrete bins to simplify the data and make it easier to analyze.

2.4 Summary of Defining Discrete

It is essential to comprehend the notion of "discrete" in the context of AI, Data Science, and Data Analytics. Discrete data is a fundamental element in these fields, and each has its own techniques for utilizing this type of data for various purposes. Knowing and managing discrete data correctly is essential for obtaining precise and useful insights.

3 Discrete Data

Discrete data consists of distinct, separate values, often derived from counting activities or non-continuous functions. Unlike continuous data, which can take an infinite number of possibilities within a given range, discrete data can only take specific, isolated values. In the contexts of AI, Data Science, and Data Analytics, discrete data can serve various purposes and can be approached in different ways. In this section discrete data will be described for the areas of AI, Data Science and Data Analytics.

3.1 Artificial Intelligence (AI) Discrete Data

In artificial intelligence, discrete data is often encountered in natural language processing, computer vision, and reinforcement learning. The usage of discrete data in AI often necessitates specialized handling, especially in terms of data preprocessing and model selection. This understanding is critical for making informed decisions in both academic and applied AI projects.

- **Tokenized Text**: Words or phrases turned into distinct integer values or vectors (word embeddings).
- **Pixels in Images**: Although the color spectrum is continuous, digital representations are discrete, typically in the range of 0-255 for each color channel.
- **Natural Language Processing (NLP)**: In NLP, text data is usually transformed into discrete tokens. Algorithms like bag-of-words or TF-IDF are based on counting the occurrence of discrete elements.
- **Reinforcement Learning**: The states, actions, and rewards are often represented as discrete variables in environments like grid worlds or game boards.
- **Action Spaces**: In many RL environments, the set of possible actions is discrete (e.g., move up, down, left, right).
- **Classification Algorithms**: Algorithms like decision trees, k-NN, and Naive Bayes often work with discrete data for class labels and sometimes for features.

3.2 Data Science Discrete Data

In the context of Data Science, discrete data is crucial for various types of analyses, modeling, and interpretation. Understanding the particularities of discrete data is essential for effective data analysis, model building, and decision-making in Data Science. The specific techniques used may differ based on the nature of the discrete data and the problem being solved.

- **Categorical Variables**: Nominal (e.g., colors, gender) or ordinal (e.g., low/medium/high) variables.
- **Counts**: Number of occurrences, such as the number of users, purchases, or clicks.
- **Time Series with Specific Intervals**: Data points collected or aggregated at regular time intervals (e.g., daily stock prices).
- **Statistical Analysis**: Descriptive statistics and hypothesis tests for discrete variables offer insights into the distribution, relationships, and structure of the data.
- **Data Visualization**: Discrete data can be effectively visualized using bar charts, histograms, or scatter plots where the x/y axis takes only discrete values.
- **Machine Learning**: In supervised learning, discrete data is commonly used for classification tasks. Decision trees and ensemble methods often handle discrete data well.

3.3 Data Analytics Discrete Data

In Data Analytics, discrete data is integral for various analytical methods, reporting frameworks, and decision-making processes. Understanding the characteristics of discrete data is essential for choosing the right analytical methods and tools in Data Analytics. Knowing how to appropriately preprocess, visualize, and model discrete data can lead to more accurate and actionable insights.

- **Key Performance Indicators (KPIs)**: Metrics like conversion rates, user counts, etc., are often discrete.
- **Survey Data**: Answers to multiple-choice questions are discrete.
- **Business Metrics**: Quarterly sales figures, monthly active users, and other metrics are usually represented as discrete data points.
- **Business Intelligence**: KPIs (Key Performance Indicators) are often discrete metrics, like the number of sales in a day or the number of clicks on an advertisement.
- **Data Aggregation**: Discrete data is often binned into categories to make it easier to analyze trends over time.
- **Time Series Analysis**: Although time can be continuous, time series data is often discretized into equally spaced intervals (e.g., daily stock prices).

3.4 One Dimensional Discrete Data

One-dimensional discrete data consists of a single set of isolated, distinct values. Unlike multi-dimensional data where each data point can have several features or attributes, one-dimensional discrete data is simple and straightforward but can still provide valuable insights. Understanding one-dimensional discrete data is essential for selecting the right analytical and statistical methods, whether you're dealing with Data Analytics, Data Science, or AI. Its simplicity allows for quick insights but also comes with limitations that one must be aware of.

3.4.1 Types of One-dimensional Discrete Data

Categorical: Data that can take on a limited number of distinct categories or states, such as "Yes" or "No," or colors like "Red," "Blue," and "Green."

Ordinal: Similar to categorical, but there's an inherent order to the categories, such as "Low," "Medium," "High."

Count: Data that represents the number of occurrences of an event, like the number of clicks, number of purchases, etc.

3.4.2 AI Applications

Text Classification: When classifying text into categories (like spam or not spam), the output is one-dimensional and discrete.

Image Classification: The same applies to image classification tasks where each image is assigned a single label from a set of discrete options.

3.4.3 Data Science and Data Analytics Applications

A/B Testing: One-dimensional discrete data like conversion rates can provide quick insights into the performance of two different web page designs, for example.

Customer Feedback: Ratings (ordinal data) from customer surveys can be analyzed to understand customer satisfaction.

Inventory Management: The number of sales each day (count data) can help businesses understand demand patterns.

3.4.4 Example - Free Spoken Digit Dataset (FSDD)

The FSDD dataset is an audio/speech dataset consisting of recordings of spoken digits in WAV files at 8kHz. The 3000 recordings, 300 of each digit, were generated from six speakers with English pronunciations (Free Spoken Digit Dataset (FSDD), 2018). In this dataset, the files contained varying initial lag times, amplitudes, and sample lengths. In document, we will use the term mic level for the silence at the beginning of each file if a mic level exists.

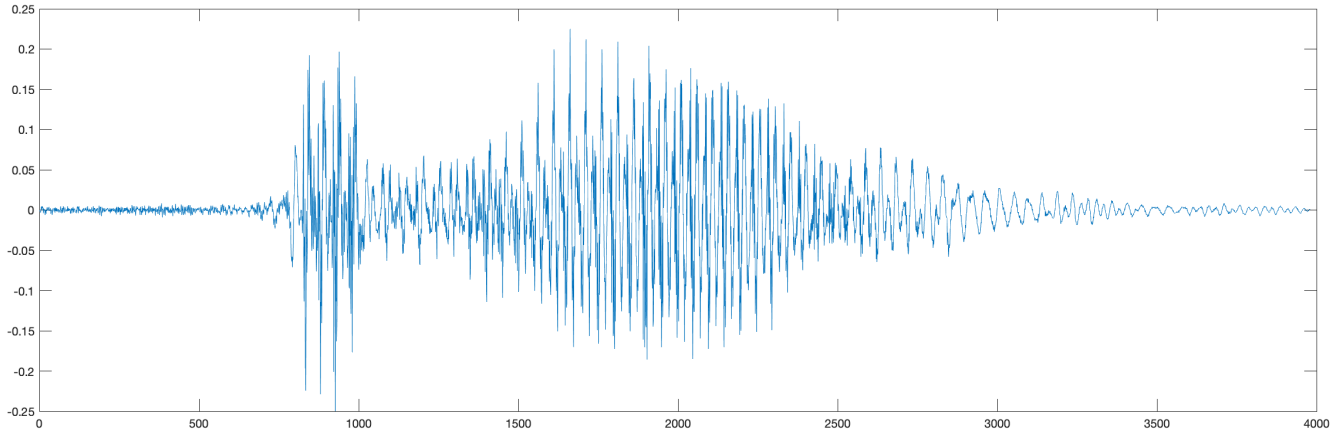


Figure 3: Original 901^{st} observation for the number 3 with mic noise lever shown for the first 783 samples and an amplitude range between -0.25 and 0.25.

Though the original files were collected/recorded at 8kHz, some files contained as little as 1000 data points in length and as long as 11000 data points. Further, the lag-time both before and after a number was spoken varied from observation to observation. Thus, the original audio files were dissimilar enough to require unique pre-processing before using combined methods. First, in the example provided here we determined the minimum and maximum volume of the file and normalized to extend the minimum to a range between .975 and -.975 amplitude. Second, we removed the noise floor at the beginning of the file by cutting off any values within the range of .18 amplitude. The normalized audio files is zero-padding to a length of 4000 total data points. Lastly, the data is base-lined to center about 0 on the y-axis. Figure 3 shows an example of the pronunciation of the number 3. The amplitude in the original file was too low, with a range between -.25 and .25. Secondly, there is a 783 sample delay before the number is spoken. In Figure 4, the same file is shown with the 783 sample lag removed and the amplitude of the entire spoken portion extended to the .975 range in amplitude. These steps enabled us to continue processing the files using transformations described in this document.

While the data looks line a continuous line in Figure 4 it is important to understand that this audio file is discrete with 8000 data points over one second of time which is known as an 8kHz data file. To further describe the discrete data points of the data in Figure 5 a zoom in from 2000 to 2050 is shown. The blue dots with red circles are shown to demonstrate the data points in the file.

3.5 Two Dimensional Discrete Data

Two-dimensional discrete data involves a dataset where each observation consists of two distinct, isolated values . This could be, for example, a set of (x, y) coordinates, or paired observations from two different categorical variables. Understanding two-dimensional discrete data is critical for selecting the appropriate analytical methods and visualization techniques. Its complexities offer both challenges and opportunities for extracting valuable insights in Data Analytics, Data Science, and AI.

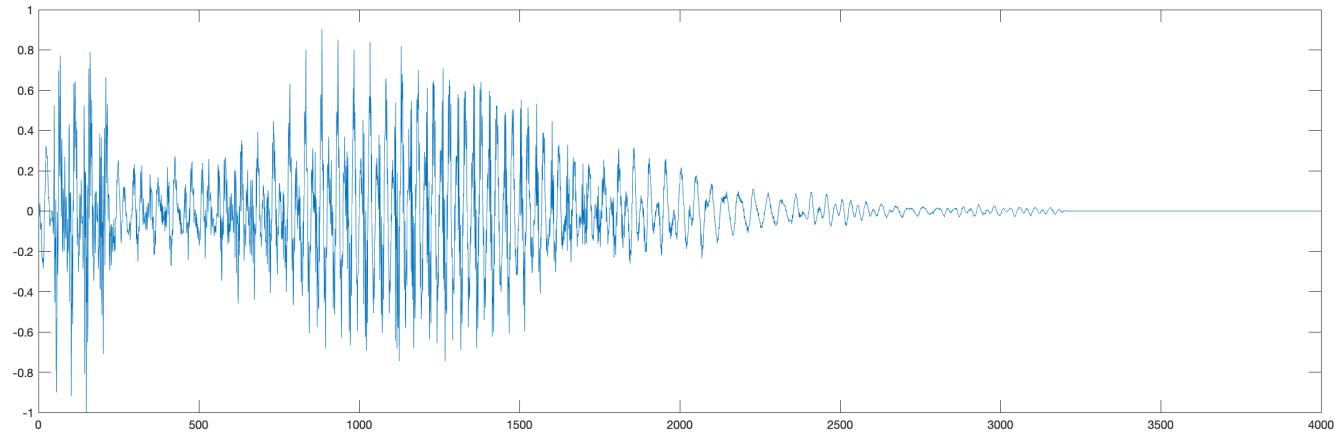


Figure 4: Processed observation using normalization, mic level removed, zero padded and re-based lined to the zero line.

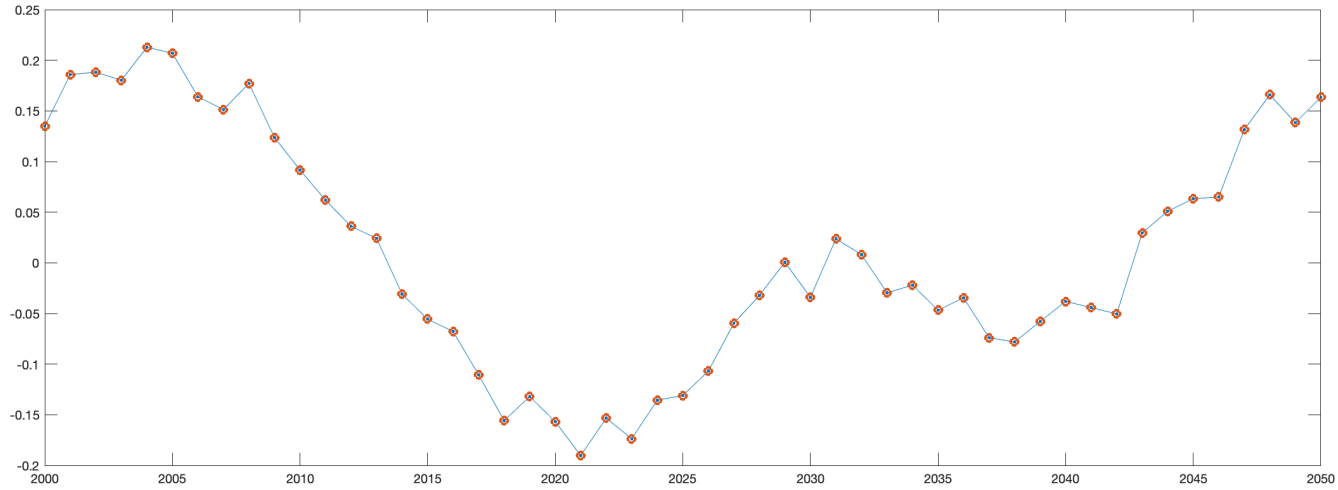


Figure 5: Zoom in for processed observation using normalization, mic level removed, zero padded and re-based lined to the zero line. from data points 2000 to 2050.

3.5.1 Types of Two-dimensional Discrete Data

Categorical-Pair: Both dimensions are categorical, like survey responses regarding brand preference and purchasing habits.

Ordinal-Pair: Both dimensions have an inherent order but are still discrete, such as school grades in two subjects.

Count-Pair: Both dimensions represent counts or frequency of occurrences, like the number of clicks and number of purchases for each user.

Mixed: One dimension is categorical, and the other is ordinal or a count, such as income level versus the number of purchases.

3.5.2 AI Applications

Multi-label Classification: Although the focus is usually on higher-dimensional data, some simplified problems might only require classifying instances into two categories, each representing one dimension.

Natural Language Processing: In sentiment analysis, you may want to classify text both by sentiment (positive, negative, neutral) and subject matter (politics, technology, sports).

3.5.3 Data Science and Data Analytics Applications

Market Basket Analysis: Understanding the relationship between two sets of items purchased together.

User Segmentation: Categorizing users based on two dimensions like engagement level and purchase history.

Risk Assessment: Assessing the risk based on two discrete variables, like credit score categories and loan repayment histories.

3.5.4 Example - MNIST Data

The MNIST data contains training.csv, test.csv and sample_submission.csv files (Modified National Institute of Standards and Technology (MNIST), 2012). In this document the focus is on the train.csv data. The data files train.csv and test.csv contain observations in rows that represent the images of hand-drawn digits, labeled from zero through nine. Each row contains 785 values with the first value being the class label and the remaining 784 values representing image pixels that can be reshaped to 28×28 matrices. There are 42000 rows representing images in the train.csv file, the only prepossessing done was converting the rows into 28×28 matrices. To show the discrete data in an image Figure 6 shows how the discrete data values can be combined to create an image.

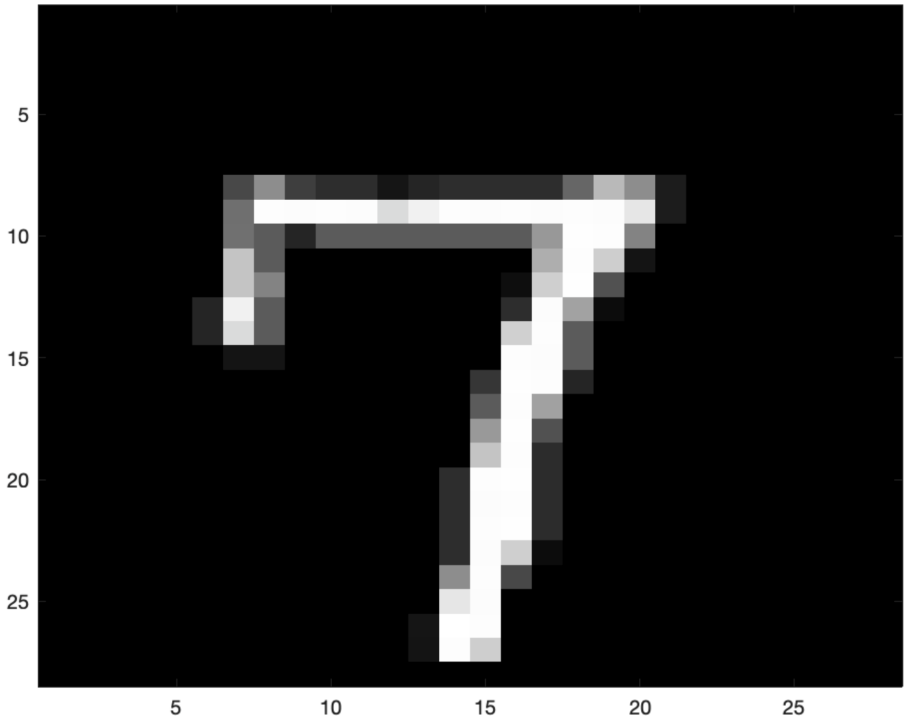


Figure 6: Numerical image 7 index 7

The values that are used to generate the image in Figure 6 are from the train.csv data index number 7. The values are shown in Figure 7. These discrete values placed in the 28×28 matrix can be processed to generate a visual representation of the image, it can also be used to process the discrete values.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	82	152	71	51	51	21	41	51	51	51	51	113	193	152	30	0	0	0	0	0	0	0
0	0	0	0	0	122	253	252	253	252	223	243	253	252	253	252	253	252	233	30	0	0	0	0	0	0	0
0	0	0	0	0	123	102	41	102	102	102	102	102	102	102	162	254	253	142	0	0	0	0	0	0	0	0
0	0	0	0	0	203	102	0	0	0	0	0	0	0	0	183	253	212	20	0	0	0	0	0	0	0	0
0	0	0	0	0	203	142	0	0	0	0	0	0	0	11	213	254	91	0	0	0	0	0	0	0	0	0
0	0	0	0	41	243	102	0	0	0	0	0	0	0	51	252	172	10	0	0	0	0	0	0	0	0	0
0	0	0	0	41	223	102	0	0	0	0	0	0	0	214	253	102	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	20	20	0	0	0	0	0	0	0	253	252	102	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	62	254	253	41	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	102	253	171	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	163	254	91	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	203	253	50	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	51	253	254	50	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	51	252	253	50	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	51	253	254	50	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	51	252	213	10	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	152	253	82	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	233	252	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	21	255	253	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	20	253	212	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7: Numerical values for the number 7 index 7

3.6 Summary of Discrete Data

The handling of discrete data often involves specific techniques for cleaning, transforming, and modeling, tailored to the nature of the discreteness. For instance, imputation strategies for missing data will differ for discrete and continuous data. In addition, discrete data consists of distinct, isolated values and can exist in various dimensions—be it one-dimensional, two-dimensional, or multi-dimensional. These data points can be categorical, ordinal, or represent counts, and they play a critical role in various fields like Data Analytics, Data Science, and AI. Understanding the nature and complexities of discrete data is crucial for selecting appropriate analytical techniques and deriving meaningful insights.

4 Discrete Mathematical Transformations

Now let's discuss how we represent raw data in a manageable form. Features/attributes where large datasets, such as images, can be exploited with a much smaller set of features. We will call this feature generation, where the basic concept of generating features is to transform a given audio, data, image, or video file which contains an extensive number of data values, into a new set of features that will allow the original data files to be represented in a much smaller set of values. It should be noted that the transform alone may not be the only data mapping step in generating the features. This will be shown in the generating features section. If the discrete mathematical transformation is suitably chosen, the transform domain values can exhibit informational properties about the original input data in a compact form. This means that most of the related information is compressed in a relatively small number of values leading to a reduced feature space (Theodoridis and Koutroumbas, 2006).

For example, consider the input image in Figure 1 and Figure 2 that is of size 512x512 pixels. This image would contain 262,144 pixel values, mapping the image into a new domain with the use of a discrete wavelet transformation can be represented as a much smaller 64x64 image with a significantly smaller number of pixels, 4096. In the new transform space the 4096 values can be used as features. The basic reasoning behind transform-based features is that an appropriate chosen transform can exploit and remove redundancies that usually exist in digital images (Theodoridis and Koutroumbas, 2006). Consider the problem of character recognition, an input image that has been taken can be represented with a much smaller set of values to determine what the actual hand written character is. Generating features for discriminating between the numbers 0 - 9 using the Discrete Cosine Transform (DCT) will eliminate redundant pixel information. When generating features derived from calculating the DCT, most of the energy lies in the frequency bands of the coefficients providing important information for class discrimination. This however leads to a large number of features, which for classification accuracy must be reduced.

The JPEG (Joint Photographic Experts Group) image compression standard is one of the most widely used image compression methods, and it employs the Discrete Cosine Transform (DCT) as a crucial component. The DCT helps transform the image from the spatial domain to the frequency domain. The DCT is crucial because it's computationally efficient and packs most of the signal information into fewer coefficients, thereby making it easier to compress without significant loss of quality. It's a form of lossy compression: you won't get the original image back, but you can get very close, depending on your quality settings. The use of DCT in JPEG compression allows for a significant reduction in file size with acceptable quality, making it ideal for various applications ranging from web images to digital photography.

The discrete mathematical transformations used in this document are the Karhunen-Loève Transform (KLT), discret wavelet transform, discrete cosine transform (DCT), and the discrete Fourier transform (DFT). While this is not a complete set of transforms, it is a good introduction into the use of discrete mathematical transformations for reducing dimensionality as well as generating features. Below are the generic forms of the transformations presented in this document. In the following section detailed information of each transformation is presented.

Karhunen-Loève Transform (KLT)

The goal of KLT is to find the orthonormal basis, $\mathbf{U}$, that diagonalizes the covariance matrix, $\mathbf{C}$, of a random vector $\mathbf{X}$.

$$\mathbf{C} = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T\right]$$

The orthonormal basis vectors, or the principal components, are the eigenvectors of the covariance matrix:

$$\mathbf{C}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

$$\mathbf{Y} = \mathbf{U}^T(\mathbf{X} - \mathbb{E}[\mathbf{X}])$$

Inverse Karhunen-Loève Transform (KLT)

$$\mathbf{X} = \mathbf{U}\mathbf{Y} + \mathbb{E}[\mathbf{X}]$$

Discrete Wavelet Transform (DWT)

This is more complex, usually represented through filter banks.

$cA[n] = \sum_k h[k] \cdot x[2n - k]$

$cD[n] = \sum_k g[k] \cdot x[2n - k]$

Inverse Discrete Wavelet Transform (IDWT)

$x[n] = \sum_m cA[m] \cdot h_r[2n - m] + \sum_m cD[m] \cdot g_r[2n - m]$

Discrete Fourier Transform (DFT)

$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j \frac{2\pi}{N} kn}$

Inverse Discrete Fourier Transform (IDFT)

$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j \frac{2\pi}{N} kn}$

Discrete Cosine Transform (DCT)

$X[k] = \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right)$

Inverse Discrete Cosine Transform (IDCT)

$x[n] = \sum_{k=0}^{N-1} X[k] \cdot \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right)$

Discrete Wavelet Transform (DWT)

This is more complex, usually represented through filter banks.

$cA[n] = \sum_k h[k] \cdot x[2n - k]$

$cD[n] = \sum_k g[k] \cdot x[2n - k]$

Inverse Discrete Wavelet Transform (IDWT)

$x[n] = \sum_m cA[m] \cdot h_r[2n - m] + \sum_m cD[m] \cdot g_r[2n - m]$

5 Eigen Decomposition

In this section the differences and uses for generating features is discussed using eigendecomposition of a matrix to represent the matrix in terms of eigenvalues and eigenvectors. This is typically conducted using Principal Component Analysis and/or the Karhunen-Loève Transform. In PCA, there are assumptions made in the derivation that must be accounted for when scaling of the variables and the applicability. We will see later that PCA can be used for dimensionality reduction, however, caution should be used to ensure information representing the data is preserved in the principal components. In the case of the Karhunen-Loève transform, the coefficients are random variables. The orthogonal basis functions in the KLT are determined by the covariance function and how it is processed. In this section, the goal is to generate features that are uncorrelated. Keep in mind that this is not a full explanation of the differences between PCA and KLT, rather an introduction is presented to ensure the use of PCA for dimensionality and KLT for generating features is introduced.

5.1 Principal Component Analysis (PCA)

The idea of feature extraction using PCA (Hotelling, 1933) is to represent a new space in a way to extract mutually uncorrelated features from the current space. The new features are known as the principal components after transform mapping. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaiser's criterion (eigenvalues: $\lambda \geq 1$) (Kaiser, 1960). The criterion is used as a guide line to determine the number of principal components to retain by calculating the correlation matrix of the input features. Each observed variable contributes one unit of variance to the total variance in the data set. Hence, any principal component that has an eigenvalue, λ greater than one accounts for a greater amount of variance than had been contributed by one variable. Additionally, a principal component that displays an eigenvalue less than one indicates less variance than had been contributed by one variable. The covariance matrix, Σ, is used to extract eigenvectors, e, retaining only the number of principal components corresponding to Kaiser's criterion.

The basic concept of feature extraction using PCA is to map x onto a new space capable of reducing the dimensionality of the input space. The data is partitioned by variance using a linear combination of 'original' factors. To perform PCA, let $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in X_n$ be a set of training vectors from the n-dimensional input space X_n. The set of vectors $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m] \in \hat{X}_m$ is a lower dimensional representation of the input training vectors $\mathbf{x}$ in the m-dimensional space $\hat{X}_m$. The vectors $\hat{x}$ are obtained by the linear orthonormal projection

$$\hat{\mathbf{x}} = \mathbf{A}^T(\mathbf{x} - \mu) \tag{1}$$

where $\mathbf{A}$ is an $[n \times m]$ matrix containing the top m eigenvectors and μ is the mean of the each set of features from $\mathbf{x}$.

5.2 Karhunen-Loève Transform (KLT)

NOTE: The below writeup is from Theodoridis and Koutroumbas, 2006, pp. 266-270.

Let

$$\hat{\mathbf{x}} = \mathbf{A}^T\mathbf{x} \tag{2}$$

From the definition of the correlation matrix we have

$$R_y \equiv E[\mathbf{y}\mathbf{y}^T] = E[\mathbf{A}^T\mathbf{x}\mathbf{x}^T\mathbf{A}] = \mathbf{A}^T R_x \mathbf{A} \tag{3}$$

Where R_x is a symmetric matrix, and hence its eigenvectors are mutually orthogonal. Thus, if the matrix A is chosen so that its columns are the orthonormal eigenvectors $\mathbf{a}_i$, $i = 0, 1, \ldots, N-1$, of R_x, then R_y is diagonal

$$R_y = \mathbf{A}^T R_x \mathbf{A} = \Lambda \tag{4}$$

where Λ is the diagonal matrix having as elements on its diagonal the respective eigenvalues λ_i, $i = 0, 1, \ldots, N-1$, of R_x. Furthermore, assuming R_x to be positive definite the eigenvalues are positive. The resulting transform is known as the Karhunen-Loève transform, and it achieves our original goal of generating mutually uncorrelated features. The KLT is of fundamental significance in pattern recognition and in a number of signal and image processing applications. Let us look at some of its important properties.

Mean square error approximation. From the definition of unitary matrices we have

$$\mathbf{x} = \sum_{i=0}^{N-1} y(i)\mathbf{a}_i \tag{5}$$

and

$$y(i) = \mathbf{a}_i^T \mathbf{x} \tag{6}$$

Let us now define a new vector in the m-dimensional subspace

$$\hat{\mathbf{x}} = \sum_{i=0}^{m-1} y(i)\mathbf{a}_i \tag{7}$$

where only m of the basis vectors are involved. Obviously, this is nothing but the projection of $\mathbf{x}$ onto the subspace spanned by the m (orthonormal) eigenvectors involved in the summation. If we try to approximate $\mathbf{x}$ by its projection $\hat{\mathbf{x}}$, the resulting mean square error is given by

$$E\big[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\big] = E\left[\left\| \sum_{i=0}^{N-1} y(i)\mathbf{a}_i \right\|^2\right] \tag{8}$$

Our goal now is to choose the eigenvectors that result in the minimization MSE. From the above equation and taking into account the orthonormality property of the eigenvectors, we have

$$E\left[\left\| \sum_{i=0}^{N-1} y(i)\mathbf{a}_i \right\|^2\right] = E\left[\sum_i \sum_j (y(i)\mathbf{a}_i^T)(y(j)\mathbf{a}_j)\right] = \sum_{i=m}^{N-1} E[y^2(i)] = \sum_{i=m}^{N-1} \mathbf{a}_i^T E[\mathbf{x}\mathbf{x}^T]\mathbf{a}_i \tag{9}$$

Combining the previous two equations and the eigenvector definition, we finally get

$$E\big[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\big] = \sum_{i=m}^{N-1} \mathbf{a}_i^T \lambda_i \mathbf{a}_i = \sum_{i=m}^{N-1} \lambda_i \tag{10}$$

Thus, if we chose in equation (7) the eigenvectors corresponding to the m largest eigenvalues of the correlation matrix, then the error in equation (10) is minimized, being the sum of the $N - m$ smallest eignevalues. Furthermore, it can be shown that this is also the minimum MSE, compared with any other approximation of x and an m-dimensional vector. This is the reason the KLT is also know as PCA.

A difference from the KLT results if we compute A in terms of the eigenvectors of the covariance matrix. This transform diagonalizes the covariance matrix Σ_y,

$$\Sigma_y = A^T \Sigma_x A = \Lambda \tag{11}$$

In general, the two are different and coincide for zero mean random vectors. In practice this is usually the case, because if it is not true one can replace each vector by $\mathbf{x} - E\mathbf{x}$. Despite that, it is still interesting to point out a difference between the two varants of the KLT. It can be shown that in this case, the resulting orthonormal basis (Σ_x eigenvectors $\hat{a}_i$) guarantees that the mean square error beetween $\mathbf{x}$ and its approximation given by

$$\hat{\mathbf{x}} = \sum_{i=0}^{m-1} y(i)\hat{\mathbf{a}}_i + \sum_{i=m}^{N-1} E\big[y(i)\hat{\mathbf{a}}_i\big], \qquad y(i) \equiv \hat{\mathbf{a}}_i^T \mathbf{x} \tag{12}$$

in minimum. In words, the last $N-m$ components are not random but are frozen to their respective mean value.

The optimality of the KLT, with respect to the MSE approximation, leads to excellent information packing properties and offers us a tool to select the m dominant features out of N measurement samples. However, although this may be a good criterion, in may cases it does not necessarily lead to maximum class separability in the lower dimensional subspace. This is reasonable, since the dimensionality reduction is not optimized with respect to class separability. This is demonstrated via the example of Figure 8. The feature vectors in the two classes follow the Gaussian distribution with the same covariance matrix.

The ellipses show the curves of constant pdf values. We have computed the eigenvectors of the overall correlation matrix, and the resulting eigenvectors are shown in the figure. Eigenvector $\mathbf{a}_i$ is the one that corresponds to the largest eigenvalue. It does not take time for someone to realize that projection on $\mathbf{a}_1$ makes the two classes almost coincide. However, projecting on $\mathbf{a}_2$ keeps the teo classes separable.

Total Variance Let $E[\mathbf{x}]$ be zero. If this is not the case, the mean can always be subtracted. Let $\mathbf{y}$ be the KLT vector of $\mathbf{x}$. From the respective definitions we have that $\sigma^2_{y(i)} \equiv E[y^2(i)] = \lambda_i$. That is, the eigenvalues of the input correlation matrix are equal matrix are equal to the variances of the transformed features. Thus, selecting those features, $y(i) \equiv \mathbf{a}_i^T\mathbf{x}$, corresponding to the m largest eigenvalues makes their sum variance $\sum_i \lambda_i$ maximum. In other words, the selected m features retain most of the total variance associated with the original random variables $x(i)$. Indeed, the latter is equal tot he trace of R_x, which we know from linear algebra to be equal to the sum of the eigenvalues $\sum_{i=0}^{N-1} \lambda_i$. It can be shown that this is a more general property. That is, from all possible sets of m features, obtained via any orthogonal linear transformation on $\mathbf{x}$, the ones resulting from the KLT have the largest sum variance.

Entropy We know that the entopy of a process is defined as

$$H_y = -E[\ln p_y(\mathbf{y})] \tag{13}$$

and it is a measure of the randomness of the process. For a zero mean Gaussian multivariable m-dimensional process the entropy becomes

$$H_y = \frac{1}{2}E[\mathbf{y}^T R_y^{-1}\mathbf{y}] + \frac{1}{2}\ln|R_y| + \frac{m}{2}\ln(2\pi) \tag{14}$$

However,

$$E[\mathbf{y}^T R_y^{-1}\mathbf{y}] = E[\text{trace}(\mathbf{y}^T R_y^{-1}\mathbf{y})] = E[\text{trace}(R_y^{-1}\mathbf{y}\mathbf{y}^T)] = \text{trace}(I) = m \tag{15}$$

and using the known property from linear algebra the determinant is

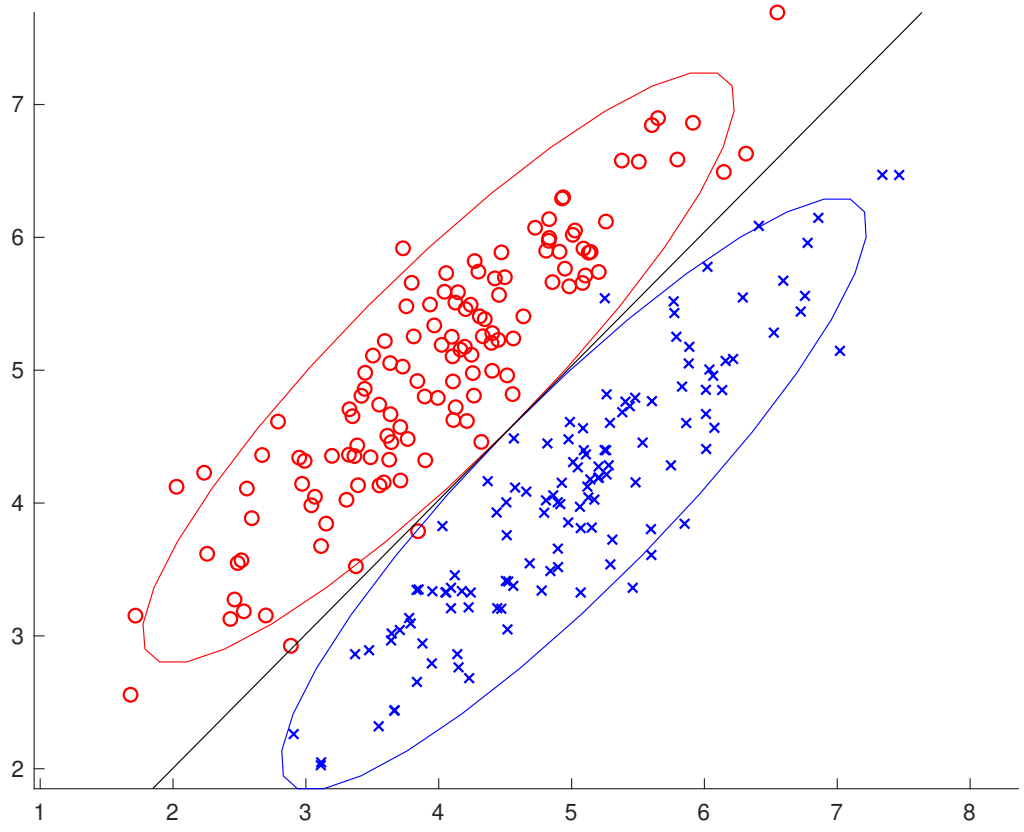
$$\ln|R_y| = \ln(\lambda_0\lambda_1 \cdot \lambda_{m-1}) \tag{16}$$

In words, selection of the m features that correspond to the m largest eigenvalues maximizes the entropy of the process. This is expected, because variance and randomness are directly related.

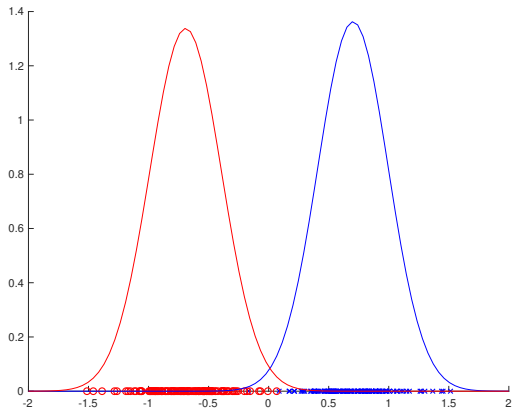
5.2.1 Karhunen-Loève Transform (KLT) Algorithm

The Karhunen-Loève Transform (KLT), also known as Principal Component Analysis (PCA) in statistics, is often used for dimensionality reduction or feature extraction. Here's the simple Algorithm 1 shown describes the KLT. The algorithm outlines the key steps: computing the mean vector, calculating the covariance matrix, finding its eigenvalues and eigenvectors, and then using these to transform the data.

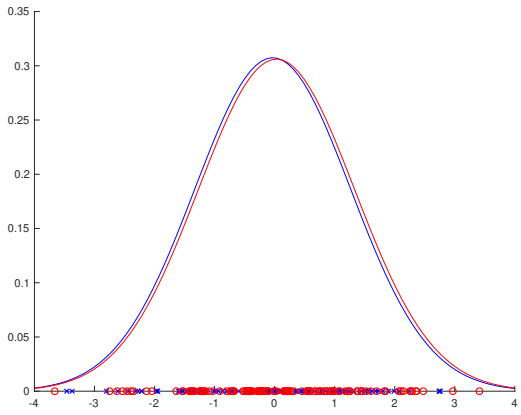
In Algorithm 1, the function KLT takes a data matrix $\mathbf{X}$ as its input. This matrix is assumed to have N samples, each of dimension d. The algorithm then returns the transformed data $\mathbf{Y}$, the eigenvectors $\mathbf{A}$, and the mean vector μ.



(a) Generated Data



(b) Smaller eigenvector projection



(c) Larger eigenvector projection

Figure 8: The KLT is not always best for pattern recognition. In this example, projection on the eigenvector with the larger eigenvalue makes the teo classes coincide. On the other hand, projection on the other eigenvector keeps the class separated.

Algorithm 1 Karhunen-Loève Transform (KLT)

function KLT($\mathbf{X}$)
  $N \leftarrow$ number of samples in $\mathbf{X}$
  $d \leftarrow$ dimensionality of each sample
  $\mu \leftarrow \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$ ▷ Compute the mean vector
  $\mathbf{C} \leftarrow \frac{1}{N}(\mathbf{X}-\mu)^T(\mathbf{X}-\mu)$ ▷ Compute the covariance matrix
  $(\mathbf{A}, \mathbf{V}) \leftarrow$ EigenDecomposition($\mathbf{C}$) ▷ Find eigenvalues and eigenvectors
  $\mathbf{A} \leftarrow$ SortByEigenvalue($\mathbf{A}, \mathbf{V}$) ▷ Sort eigenvectors by eigenvalues
  $\mathbf{Y} \leftarrow (\mathbf{X}-\mu)\mathbf{A}$ ▷ Transform the data
  return $\mathbf{Y}, \mathbf{A}, \mu$ ▷ Return transformed data, eigenvectors, and mean vector
end function

5.3 Number of Components/Eigenvectors to Retain (Dillon and Goldstein, 1984)

Principal component analysis is frequently employed for the purpose of generating a reduces set of variates that account for most of the variability in the original data, and that can be used in more substantial subsequent analysis. WE must therefore decide just how many components to retain. Unfortunately there is not universally accepted method for doing so. The decision is largely judgmental and a matter of taste.

A number of procedures for determining how many components to retain have been suggested. These "rules" range from methods that evoke formal significance tests to less formal approaches involving heuristic graphical arguments. The remainder of this section discusses several of the more commonly used procedures.

Variance - Covariance Input A reasonable approach for determining the number oof components to retain when factoring a variance-covariance matrix relies on the sampling distribution results. For example, we might suggest that only those components whose associated eigenvalues are statistically different from zero be retained.

The reader should note, however, that with even moderate sample sizes many of the components will typically be statistically significant. Yet, from a practical viewpoint, some of these significant components account for only a vary small proportion of the total variance. Hence, for reasons of parsimony and practical significance, the statistical criteria should be interpreted loosely with fewer components retained than are statistically significant. In this regard,, some of the graphical procedures soon to be discussed may prove useful.

An alternative and somewhat more ad hoc approach is the percentage of variance criterion. With this approach, the cumulative percentage of eh variance extracted by successive components is the criterion. Note, the cumulative proportion of total variance extracted by a set of components sis given by

$$\frac{\sum_{j=}^{m} l_{(j)}}{\sum_{j=}^{p} l_{(j)}} \tag{17}$$

where $m < p$. The stopping rule is subjective, since the number of components retained is based on some arbitrary determined criterion for the amount of variation accounted for.

Correlation Input When factoring a correlation matrix, statistical testing procedures no longer apply, and the retained variance criterion lacks clear meaning. For these reasons, various graphical heuristics have been suggested as a rule of thumb.

Perhaps the most frequent used extraction approach is the "root greater than one" criterion. Originally suggested by Kaiser (1958), this criterion retains those components whose eigenvalues are greater than one. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaiser's criterion (eigenvalues: $\lambda \geq 1$). Principal component analysis partitions the data by variance using linear combination of 'original' factors. The rationale for this criterion is

that any component should account for more "variance" than any single variable in the standardized test score space.

Another approach, proposed by Cattell (1966), is called the scree test. With this approach, named after the rubble at the bottom of a cliff, the eigenvalues of each component are plotted in successive order of their extraction, and then an elbow in the curve is identified by applying, say, a straightedge to the bottom portion of the eigenvalues to see where they form an approximate straight line. The number of components retained is given by the point at which the components curve above the straight line formed by the smaller eigenvalues. Cattell and Jaspers (1967) suggested that the number of factors be taken as the number immediately before the straight line begins. Figure 9 shows a hypothetical case which four components would be retained. Note that the graph clearly depicts the scree analogy - the first few eigenvalues show the cliff and the rest the rubble. The rationale for the scree test is simple: since the principal component solution extracts components in successive order of magnitude, the substantive factors appear first, followed by the numerous trivial components which account for only a small proportion of the total variance.

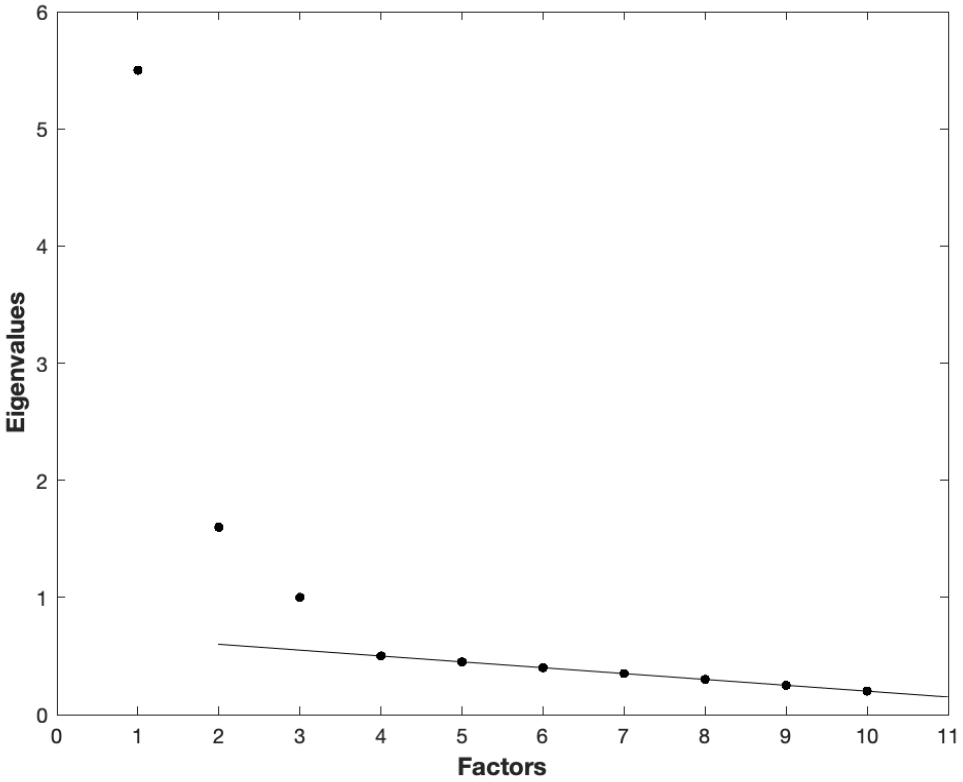


Figure 9: Illustration of the scree test.

Though this approach is relatively simple, complications can surface. First, there may be no obvious break, in which case the test is inclusive. Second, There may be several breaks. This is particularly troublesome when, say, two breaks occur among the first half of the eigenvalues, since it will be difficult to decide which of the breaks reflects the correct number of components.

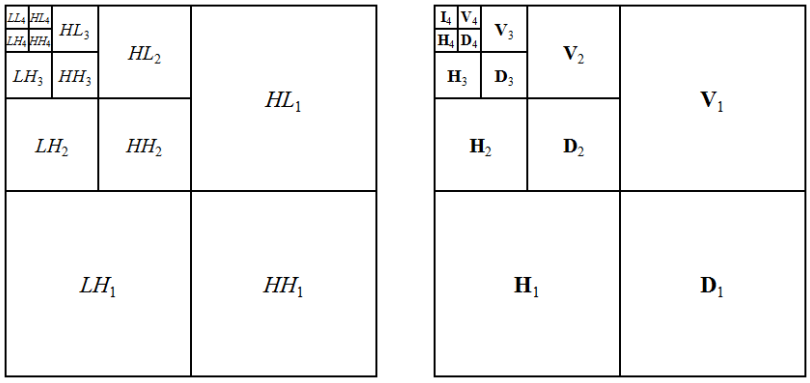
Horn (1965) has suggested yet another approach for determining the number of components to retain which removes much of the ambiguity associated with the scree test. As in the scree test, the data are factored and the resulting component eigenvalues are plotted in successive order of their magnitude. Next, K setts of $n \times p$ normally

and independently distributed (NID) random variates are sampled from a population whose correlation structure is known to be characterized by an identity matrix. Each $n \times p$ data matrix is factored. The average eigenvalue for each extracted component over the k sets of randomly generated data s then computed and plotted in the same graph with the real data. Because of sampling variation, several of the extracted eigenvalues may exceed unity. However, the average curve of eighenvalues can be expressed to cross the ordinate scale at the value 1.0 for component number $p/2$. The principal component solution for the simulated data represents the case in which the eigenvalues under the null hypothesis are unity. Consequently, Horn's criterion is to retain components n the basis of where the reference curve crosses the curve induced from the actual data. **Figure xx** shows a hypothetical application of Horn's criterion based on $p = 30$ variables.

Great Online Resource: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html

6 Wavelets

The wavelet transform is a great method for compressing image and audio files. It is also a great for identifying image vertical, diagonal and horizontal characteristics. Research has been employed using wavelet transforms as a preprocess for deriving image features. The image decomposition employed here is based on separable quadrature mirror filters (QMF) originally used in (Farid, 2002). Applying the wavelet transform on a given image maps the image pixel from the spatial domain to the wavelet transform domain, i.e., to the frequency space. The image decomposition splits the frequency space into multiple orientations and levels. For each level, four subbands, LL, LH, HL, and HH, are created inheriting the characteristics of the image, identical (**I**), vertical (**V**), horizontal (**H**) and diagonal (**D**) information, correspondingly. The symbol L is denoted as a lowpass filter being used, while H would be the case of a highpass filter (Nievergelt, 1999).



(a) Subband information (b) Edge information

Figure 10: A four-scale wavelet decomposition

The number of iterations applied, also named as a scale or level, is denoted by a subscript i.

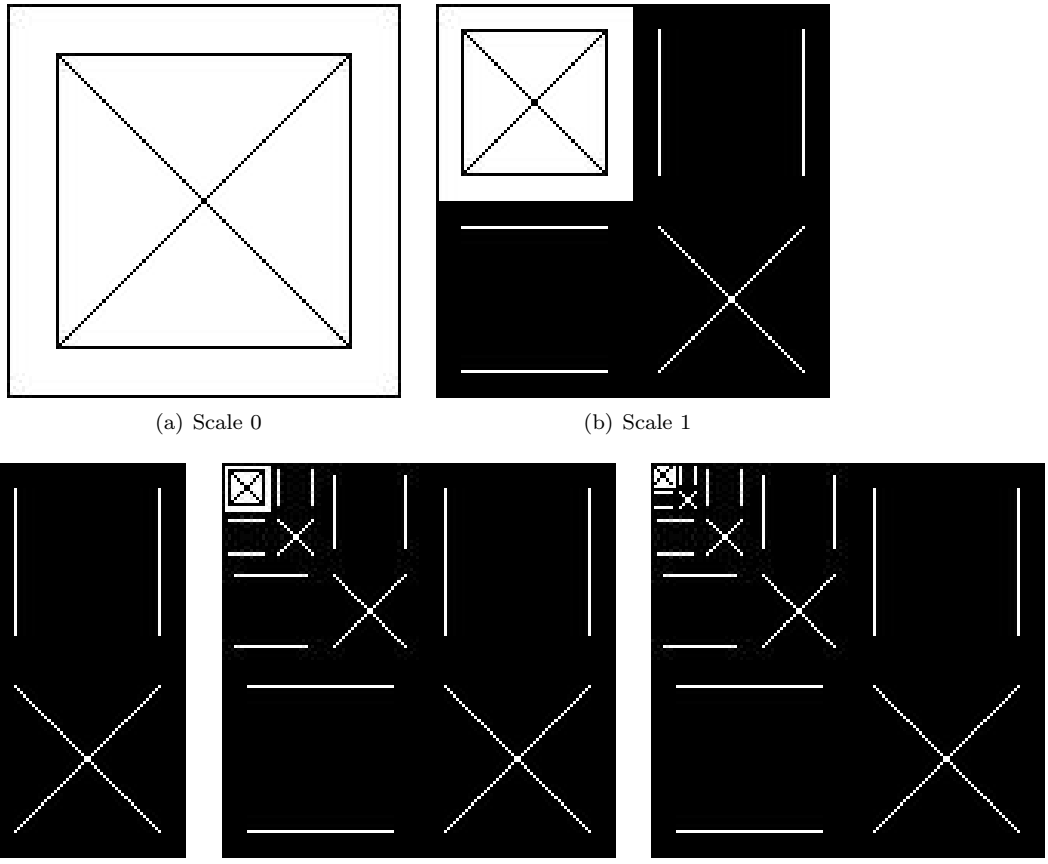
Recursively applying QMF to LL_i, Figure 10 presents an i^{th}-scale wavelet decomposition, where $i = 1, 2, 3, 4$. Suppose, there is a set of n images. For each image, a four-scale three-orientation QMF is utilized. A coefficient located within a wavelet transformed image is denoted by its edge characteristics and its position, i.e., $\mathbf{V}_i(n_1, n_2)$, $\mathbf{H}_i(n_1, n_2)$, $\mathbf{D}_i(n_1, n_2)$, where $i = 1, 2, 3, 4$, and (n_1, n_2) are the row and column index within a characteristic block. The coordinate (n_1, n_2) denotes the coefficient location within a block. Given an example image with vertical, horizontal, and diagonal lines as in Figure 11(a), Figure 11(b) to 11(e) demonstrate the wavelet decomposition results for each iteration from 1 to 4. The various scales of the wavelet coefficients are normalized for visualization purposes in Figure 11.

6.1 2D Discrete Wavelet Transform

The 2D Discrete Wavelet Transform (2D-DWT) is an extension of the 1D Discrete Wavelet Transform, widely used in image and video compression, denoising, and other signal processing applications. The 2D-DWT captures both low-frequency and high-frequency components of an image, dividing it into different subbands that can be analyzed or processed separately (Jense and la Cour-Harbo, 2001).

Mathematical Formulation

The 2D-DWT involves applying a 1D-DWT along the rows and then along the columns (or vice versa, the order is not crucial). For a 2D signal $Im(n_1, n_2)$, the 2D-DWT can be computed using a set of wavelet and scaling



(a) Scale 0 (b) Scale 1

(c) Scale 2 (d) Scale 3 (e) Scale 4

Figure 11: Wavelet decompositions on an example image

functions $\psi(n)$ and $\phi(n)$.

The first step is usually to divide the image $Im(n_1, n_2)$ into four subbands:

- LL (Low-Low): Low-frequency components in both n_1 and n_2
- LH (Low-High): Low-frequency in n_1 but high-frequency in n_2
- HL (High-Low): High-frequency in n_1 but low-frequency in n_2
- HH (High-High):: High-frequency components in both n_1 and n_2

The subbands are obtained by convolving $Im(n_1, n_2)$ with low-pass and high-pass filters followed by down-sampling.

The operation to generate a subband (for instance, LL) can be represented mathematically as:

$$\text{LL}[k_1, k_2] = \sum_{n_1} \sum_{n_2} Im[n_1, n_2] \cdot \phi(k_1 - 2n_1) \cdot \phi(k_1 - 2n_2) \tag{18}$$

For the LH subband:

$$\text{LH}[k_1, k_2] = \sum_{n_1} \sum_{n_2} Im[n_1, n_2] \cdot \phi(k_1 - 2n_1) \cdot \psi(k_2 - 2n_2) \tag{19}$$

For the HL (High frequency in n_1, Low frequency in n_2) subband:

$$\text{HL}[k_1, k_2] = \sum_{n_1} \sum_{n_2} Im[n_1, n_2] \cdot \psi(k_1 - 2n_1) \cdot \phi(k_2 - 2n_2) \tag{20}$$

Similarly, for the HH (High frequency in n_1, High frequency in n_2) subband:

$$\text{HH}[k_1, k_2] = \sum_{n_1} \sum_{n_2} Im[n_1, n_2] \cdot \psi(k_1 - 2n_1) \cdot \psi(k_2 - 2n_2) \tag{21}$$

These subbands capture the low and high-frequency information in different dimensions, which is useful for various applications such as image compression, feature extraction, and denoising.

Inverse 2D-DWT

The inverse transform can reconstruct the original signal $Im(n_1, n_2)$ from its subbands. This involves up-sampling followed by filtering using the synthesis wavelet and scaling functions.

The inverse 2D-DWT can be represented as:

$$Im(n_1, n_2) = \sum_{m} \sum_{n} \text{LL}[m, n] \cdot \phi(2m - x) \cdot \phi(2n - y) + \ldots \text{(other terms from LH, HL, HH)} \tag{22}$$

Scaling Function (ϕ)

The scaling function, often denoted by $\phi(n)$ (or $\phi(n_1, n_2)$ in 2D), serves to capture the coarse or low-frequency details of the signal. It's usually a smooth, localized function. In the realm of wavelets, the most commonly used scaling function is associated with the Daubechies wavelet family, but other families like Haar, Coiflets, and Symlets also provide their own scaling functions.

$$\phi(n) = \sum_k h[k]\sqrt{2}\phi(2n - k) \tag{23}$$

Wavelet Function (ψ)

The wavelet function, often denoted by $\psi(n)$ (or $\psi(n_1, n_2)$ in 2D), captures the high-frequency details or changes in the signal. This function is also localized and usually orthogonal to its shifted versions. The wavelet function is crafted to provide a high degree of decorrelation, making it useful for tasks like compression and denoising.

$$\psi(n) = \sum_k g[k]\sqrt{2}\phi(2n - k) \tag{24}$$

Here $h[k]$ and $g[k]$ are the low-pass and high-pass filter coefficients, respectively.

In the 2D-DWT, both ϕ and ψ are used to process the signal in both the n_1 and n_2 dimensions. This results in four subbands:

- LL (Low-Low): Uses $\phi(n_1)$ for rows and $\phi(n_2)$ for columns
- LH (Low-High): Uses $\phi(n_1)$ for rows and $\psi(n_2)$ for columns
- HL (High-Low): Uses $\psi(n_1)$ for rows and $\phi(n_2)$ for columns
- HH (High-High): Uses $\psi(n_1)$ for rows and $\psi(n_2)$ for columns

Each of these subbands captures different features of the image, making the 2D-DWT a powerful tool for image processing tasks.

6.1.1 2D Discrete Wavelet Transform Algorithm

A simple algorithm for computing the 2D Discrete Wavelet Transform (2D-DWT) with four subbands: LL, LH, HL, and HH is described and illustrated. The algorithm performs a 1D-DWT on each row, followed by a 1D-DWT on each column of the intermediate results.

In this example, Im represents the 2D array (or image) you're working with, while ϕ and ψ are the scaling and wavelet functions, respectively. The arrays LL, LH, HL, HH store the coefficients of the four subbands.

Remember that this is a simplified and abstract representation. In a real implementation, the actual steps to compute the 1D-DWT for rows and columns would be more detailed as in the equations for LL, LH, HL, and HH.

Algorithm 2 2D Discrete Wavelet Transform

function 2D-DWT(Im, ϕ, ψ)
  Initialize empty matrices: LL, LH, HL, HH
  $N_1, N_2 \leftarrow$ dimensions of Im
  for $n_1 = 0$ to $(N_1/2) - 1$ **do**
    for $n_2 = 0$ to $(N_2/2) - 1$ **do**
      $LL[n_1, n_2] \leftarrow 0$
      $LH[n_1, n_2] \leftarrow 0$
      $HL[n_1, n_2] \leftarrow 0$
      $HH[n_1, n_2] \leftarrow 0$
    end for
  end for
  for $n_1 = 0$ to $(N_1/2) - 1$ **do**
    Compute 1D-DWT of row $Im[n_1, :]$ to get L_{row}, H_{row}
    for $n_2 = 0$ to $(N_2/2) - 1$ **do**
      Compute 1D-DWT of column $L_{row}[:, n_2]$ to get LL_{col}, LH_{col}
      Compute 1D-DWT of column $H_{row}[:, n_2]$ to get HL_{col}, HH_{col}
      $LL[n_1, n_2] \leftarrow LL_{col}$
      $LH[n_1, n_2] \leftarrow LH_{col}$
      $HL[n_1, n_2] \leftarrow HL_{col}$
      $HH[n_1, n_2] \leftarrow HH_{col}$
    end for
  end for
  return LL, LH, HL, HH
end function

6.1.2 2D Inverse Discrete Wavelet Transform Algorithm

The 2D Inverse Discrete Wavelet Transform (2D-InvDWT) algorithm takes the four subbands (to LL, LH, HL, HH) as inputs and reconstructs the original image.

In this example, LL, LH, HL, HH are the four subbands obtained after the 2D-DWT, and ϕ and ψ are the scaling and wavelet functions, respectively. The function TwoDIDWT takes these as inputs and reconstructs the original 2D array (or image) Im.

As before, this is a simplified and abstract algorithm. A complete implementation would include details about the specific wavelet and scaling functions, as well as how the 1D-IDWT is computed.

Algorithm 3 2D Inverse Discrete Wavelet Transform

function 2D-InvDWT($LL, LH, HL, HH, \phi, \psi$)
    Initialize empty matrix Im
    $N_1, N_2 \leftarrow$ dimensions of LL
    for $n_1 = 0$ to $N_1 - 1$ **do**
        for $n_2 = 0$ to $N_2 - 1$ **do**
            $Im[n_1, n_2] \leftarrow 0$
        end for
    end for
    for $n_1 = 0$ to $N_1 - 1$ **do**
        Compute 1D-IDWT of columns $LL[n_1, :], LH[n_1, :]$ to get L_{col}
        Compute 1D-IDWT of columns $HL[n_1, :], HH[n_1, :]$ to get H_{col}
        for $n_2 = 0$ to $N_2 - 1$ **do**
            Compute 1D-IDWT of row $L_{col}[:, n_2], H_{col}[:, n_2]$ to reconstruct Im_{row}
            $Im[n_1, n_2] \leftarrow Im_{row}$
        end for
    end for
     return Im
end function

6.2 Haar Wavelet

The Haar wavelet is one of the simplest and earliest wavelets. It has a piecewise constant representation and is useful for its simplicity and ease of computation. The Haar wavelet is defined by its scaling function $\phi(x)$ and wavelet function $\psi(x)$.

The Haar scaling function $\phi(x)$ is defined as:

$$\phi(x) = \begin{cases} 1, & \text{if } 0 \geq x < 1 \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

The Haar wavelet function $\psi(x)$ is defined as:

$$\psi(x) = \begin{cases} 1, & \text{if } 0 \geq x < 1/2 \\ -1, & \text{if } 1/2 \geq x < 1 \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

The Haar wavelet transformation employs two sets of coefficients: one for the scaling function and one for the wavelet function. These coefficients are used in filter banks for both the forward and inverse transformations.

$$\text{Low-pass filter coefficients (1D): } h = \left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right] \tag{27}$$

$$\text{High-pass filter coefficients (1D): } g = \left[\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right] \tag{28}$$

The forward Haar wavelet transform involves convolving these coefficients with the input signal, followed by down-sampling. The inverse Haar wavelet transform uses the same coefficients but involves up-sampling followed by convolution to reconstruct the original signal.

Because of its simplicity, the Haar wavelet is often used in educational settings and for simple wavelet analysis tasks. It's not as efficient as other wavelets like Daubechies or Coiflets for most practical applications, but it provides a good introduction to the subject.

6.2.1 2D Haar Wavelet Transform

The 2D Haar Wavelet Transform is an extension of the 1D Haar Wavelet Transform to two dimensions. It is primarily used in image processing for tasks like compression, denoising, and feature extraction. Below, I'll describe how the 2D Haar Wavelet Transform works and present its four subbands.

In a 2D context, we apply these filters in both the row and column directions. The 1D Haar transform is first applied to each row of the 2D data (usually an image). This yields two sets of coefficients: approximation coefficients (low-frequency) and detail coefficients (high-frequency). Next, the same 1D Haar transform is then applied to each column of the approximation and detail coefficients. As a result, you will get four different sets of coefficients which correspond to the four subbands. To denote the filter applied along the rows and columns, we use the following notations:

- LL subband (2D): $h \otimes h$ for the low-pass filter in both dimensions, resulting in the LL subband. Contains the low-frequency components along both dimensions. This is the approximation of the original image.
- LH subband (2D): $h \otimes g$ for the low-pass filter in rows and high-pass in columns, resulting in the LH subband. Contains the low-frequency components along rows and high-frequency components along columns.

- HL subband (2D): $g \otimes h$ for the high-pass filter in rows and low-pass in columns, resulting in the HL subband. Contains the high-frequency components along rows and low-frequency components along columns.
- HH subband (2D): $g \otimes g$ for the high-pass filter in both dimensions, resulting in the HH subband. Contains the high-frequency components along both dimensions.

The above description provides the mathematical description of how the scaling functions, wavelet functions, and subbands. Each of the above descriptions can be used with matrix operations. The following is a mathematical description of the subbands and the necessary calculations.

Let Im represent an image to be transformed with the number of rows and column represented as N_1 and N_2 respectively. Now represent a 2×2 area of the image as follows:

$$\begin{pmatrix} Im_{00} & Im_{01} \\ Im_{10} & Im_{11} \end{pmatrix} \tag{29}$$

This then leads to the mathematical operation of the transforms for the subbands as follows:

- LL (Low-Low): Low-frequency components $\frac{1}{2}\left[\frac{Im_{00}+Im_{01}}{2} + \frac{Im_{10}+Im_{11}}{2}\right]$ which represents the average of the 4 values.
- LH (Low-High): Low-frequency components $\frac{1}{2}\left[\frac{Im_{00}-Im_{01}}{2} + \frac{Im_{10}-Im_{11}}{2}\right]$ which represents the average horizontal change from the first to the second column.
- HL (High-Low): High-frequency components $\frac{1}{2}\left[\frac{Im_{00}+Im_{01}}{2} - \frac{Im_{10}+Im_{11}}{2}\right]$ which represents the average vertical change from the first to the second rows.
- HH (High-High):: High-frequency components $\frac{1}{2}\left[\frac{Im_{00}-Im_{01}}{2} - \frac{Im_{10}-Im_{11}}{2}\right]$ which represents the average diagonal change.

The Haar wavelet, because of its simplicity, serves as a good introduction for those new to wavelets or signal processing.

6.2.2 2D Haar Wavelet Transform Algorithm

When dealing with a 2D matrix Im of size $N_1 \times N_2$ where the number of rows N_1 and the number of columns N_2 are not the same. Below is a representation of the 2D Haar Wavelet Transform Algorithm 4. In this representation, the algorithm separates the data into four subbands (LL, LH, HL, HH) during each level of decomposition.

In this algorithm, the function TwoDHaarWaveletTransform takes a 2D array Im and the number of decomposition levels L as its parameters. Then, for each level of decomposition, the function calculates the LL, LH, HL, HH coefficients and stores them in separate matrices. These coefficients are then copied back into a 2D matrix $waveletIm$ in their respective quadrants. The number of levels for the wavelet are accounted for and the tansformed matrix $waveletIm$ is returned.

6.2.3 2D Inverse Haar Wavelet Transform Algorithm

Algorithm 5 represents the 2D Inverse Haar Wavelet Transform algorithm. The input is the $waveletIm$ containing the subbands LL, LH, HL, and HH as well as the levels L. This algorithm works in reverse to reconstruct the original matrix Im.

In this algorithm, the function TwoDInverseHaarWaveletTransform takes the LL, LH, HL, and HH subbands along with the number of decomposition levels L as its parameters. It then proceeds to combine these subbands to reconstruct the original 2D array Im.

Algorithm 4 2D Haar Wavelet Transform

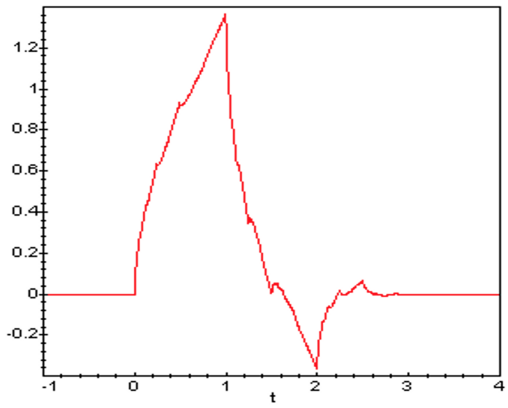
2D matrix Im of size $N_1 \times N_2$ to be transformed, number of levels L
function TwoDHaarWaveletTransform(Im, L)
$N_1 \leftarrow$ rows of Im
$N_2 \leftarrow$ columns of Im
$LL, LH, HL, HH \leftarrow$ initialize empty matrices
for $l = 1, 2, \ldots, L$ **do**
$n_1 \leftarrow \lfloor N_1/2^{l-1} \rfloor$
$n_2 \leftarrow \lfloor N_2/2^{l-1} \rfloor$
for $i = 1, 2, \ldots, n_1, 2$ **do**
for $j = 1, 2, \ldots, n_2, 2$ **do**
$im_{00} \leftarrow Im[i, j]$
$im_{01} \leftarrow Im[i, j+1]$
$im_{10} \leftarrow Im[i+1, j]$
$im_{11} \leftarrow Im[i+1, j+1]$
$LL[i/2, j/2] \leftarrow \frac{im_{00}+im_{01}+im_{10}+im_{11}}{4}$
$LH[i/2, j/2] \leftarrow \frac{im_{00}+im_{01}-im_{10}-im_{11}}{4}$
$HL[i/2, j/2] \leftarrow \frac{im_{00}-im_{01}+im_{10}-im_{11}}{4}$
$HH[i/2, j/2] \leftarrow \frac{im_{00}-im_{01}-im_{10}+im_{11}}{4}$
end for
end for
▷ Update the synthetic image $synIm$ with new subbands
$synIm[1 : n_1/2, 1 : n_2/2] \leftarrow LL$
$synIm[1 : n_1/2, n_2/2 + 1 : n_2] \leftarrow LH$
$synIm[n_1/2 + 1 : n_1, 1 : n_2/2] \leftarrow HL$
$synIm[n_1/2 + 1 : n_1, n_2/2 + 1 : n_2] \leftarrow HH$
▷ Update the image Im with LL subbands
if $l = 1$ **then**
$waveletIM \leftarrow synIm$
else
$waveletIm[0 \text{ to } N_2/2][0 \text{ to } N_2/2] \leftarrow synIm$
end if
end for
return $waveletIm$
end function

Algorithm 5 2D Inverse Haar Wavelet Transform

function TwoDInverseHaarWaveletTransform($waveletIm, L$)
  $N_1 \leftarrow$ the number of rows in $waveletIm$
  $N_2 \leftarrow$ the number of columns in $waveletIm$
  $n_1 \leftarrow N_1/(2^L)$
  $n_2 \leftarrow N_2/(2^L)$
  $LL \leftarrow waveletIm$ with the subband at the highest level
  for $l = L, L-1, \ldots, 1$ **do**
    $LH \leftarrow waveletIm(n_1 + 1 : n_1 * 2, 1 : n_2)$
    $HL \leftarrow waveletIm(1 : n_1, n_2 + 1 : n_2 * 2)$
    $HH \leftarrow waveletIm(n_1 + 1 : n_1 * 2, n_2 + 1 : n_2 * 2)$
    $N_1 \leftarrow$ the number of rows in LL
    $N_2 \leftarrow$ the number of columns in LL
    $n_1 \leftarrow N_1 * 2$
    $n_2 \leftarrow N_2 * 2$
    ▷ Initialize Im with zeros the size of n_1 and n_2
    for $i = 1, 2, \ldots, n_1, 2$ **do**
      for $j = 1, 2, \ldots, n_2, 2$ **do**
        $im_{00} \leftarrow LL[i/2, j/2]$
        $im_{01} \leftarrow LH[i/2, j/2]$
        $im_{10} \leftarrow HL[i/2, j/2]$
        $im_{11} \leftarrow HH[i/2, j/2]$
        $Im[i, j] \leftarrow im_{00} + im_{01} + im_{10} + im_{11}$
        $Im[i, j+1] \leftarrow im_{00} + im_{01} - im_{10} - im_{11}$
        $Im[i+1, j] \leftarrow im_{00} - im_{01} + im_{10} - im_{11}$
        $Im[i+1, j+1] \leftarrow im_{00} - im_{01} - im_{10} + im_{11}$
      end for
    end for
    $LL \leftarrow Im[1 : n_1, 1 : n_2]$
    ▷ Update LL for the next level
  end for
  $reconstructedIm \leftarrow Im$
  return $reconstructedIm$
end function

6.3 Daubechies Wavelet

In this subsection we will consider wavelet families, which were discovered by Ingrid Daubechies in 1988 (Daubechies, 1992). Daubechies family of wavelets includes the Haar wavelet. Daubechies required that her wavelets have compact support, which means that the member of this family has finite number of scaling coefficients and they have N/2 vanishing moments. She made a great input on the development of the wavelet theory.



(a) Daubechies 4 Wavelet

(b) Ingrid Daubechies

Figure 12: Daubechies Wavelet

The Daubechies method is based on the solution of the scaling equation

$$\phi(x) = \sum_{k=0}^{2m-1} c_k \phi(2x-k) \tag{30}$$

Where

$$\phi_0(x) = \begin{cases} 1 & if \quad 0 \leq x \leq 1 \\ 0 & otherwice \end{cases} \tag{31}$$

Finding the scaling coefficients $\{c_0, c_2, c_3, ...c_{2m-1}\}$ is not a simple task. Daubechies determined the scaling coefficients $\{c_0, c_2, c_3, ...c_{2m-1}\}$ by imposing constraints on the scaling coefficients and additional explicit requirements (see the fired equation of the system 32) on the wavelets. Then, using the recursive construction algorithm, she developed a plot of the scaling function.

The additional Daubechies conditions are:

a) The scaling function $\phi(x)$ is a nonzero in the interval $0 \leq x \leq 2m-1$.

b) Motivated by desired wavelet properties, the scaling coefficients satisfy the following system of equations:

$$\begin{cases} 1. \sum_{k=0}^{N-1} c_k = \sqrt{2} \\ 2. \sum_k c_k c_{k-2m} = 2\delta_{0m} \\ 3. \sum_{q=0}^{N-1} (-1)^q c_k q^m = 0, \quad m = 0, 1, 2, ..., p-1 \end{cases} \tag{32}$$

We may write the wavelet function ψ(x) associated (see equation 33) with the scaling function of the wavelet system as

$$\psi(x) = \sum_{k=0}^{2m-1} b_k \phi(2x-k) \tag{33}$$

$$b_k = (-1)^k c_{2m-1-k}, \qquad k = 0, 1, ..., 2m-1, \tag{34}$$

In general, the coefficients for the discrete wavelet system D_{2m}, are determined by 2m equations in 2m unknowns. They are non zero in the interval $0 \leq x \leq 2m-1$. To illustrate how the scaling coefficients are derived, consider the cases m=1,2, and 3.

Case 1. m=1, the Lenth-2 Daubechies or Haar scaling function

$$\phi(x) = \sum_{k=0}^{2m-1} c_k \phi(2x-k) = c_0\phi(2x) + c_1\phi(2x-1) \tag{35}$$

The conditions on the scaling coefficients are:

$$\begin{cases} c_0 + c_1 = \sqrt{2} \\ c_0^2 + c_1^2 = 1 \end{cases} \tag{36}$$

It easy to check that $c_0 = c_1 = 1/\sqrt{2} = 0.7071067812$ satisfies this condition. Daubechies-2 wavelet coefficients are therefore $b_0 = c_1$ and $b_2 = -c_0$. The Haar wavelet has a support length of 1.

Case 2. m=2, the Lenth-4 Daubechies scaling function. The third Daubechies condition equation applies in this case. Requiring ϕ(x) to be zero outside of the interval $0 \leq t \leq 3$ for this case satisfies the first condition; that the scaling function have compact support. Consequently, all of the scaling filter coefficients need to be zero except c_0, c_1, c_2 and c_3. We therefore obtain:

$$\phi(x) = c_0\phi(2x) + c_1\phi(2x-1) + c_2\phi(2x-2) + c_3\phi(2x-3) \tag{37}$$

Since nearly all of the scaling coefficients are zero, the second Daubechies condition, orthogonality, yields two equations.

$$c_0^2 + c_1^2 + c_2^2 + c_3^2 = 0 \tag{38}$$

$$c_0c_2 + c_1c_3 = 0 \tag{39}$$

The third condition, regularity, characterizes the smoothness of the scaling function. The motivation is that smother scaling functions generate wavelet families that can better approximate polynomials. These moment conditions $\sum_k (-1)^k\ c_k\ k^m = 0$ lead to two more equations:

$$c_3 - c_2 + c_1 - c_0 = 0 \qquad (m=1) \tag{40}$$

$$0c_3 - 1c_1 + 2c_1 + 3c_0 = 0 \qquad (m=2) \tag{41}$$

We now have four equations in four unknowns

$$c_0^2 + c_1^2 + c_1^2 + c_2^2 = 1 \tag{42}$$

$$c_0c_2 + c_1c_3 = 0 \tag{43}$$

$$c_3 - c_2 + c_1 - c_0 = 0 \tag{44}$$

$$0c_3 - 1c_1 + 2c_1 + 3c_0 = 0 \tag{45}$$

These equations are sufficient for uniquely determining the scaling coefficients to within a factor of -1. Solving these equations, we obtain Daubechies-4 scaling coefficients:

$$c_0 = (1 + \sqrt{3})/4 = 0.6830127 \tag{46}$$

$$c_1 = (3 + \sqrt{3})/4 = 1.1830127 \tag{47}$$

$$c_2 = (3 - \sqrt{3})/4 = 0.3169873 \tag{48}$$

$$c_3 = (1 - \sqrt{3})/4 = -0.1830127 \tag{49}$$

Daubechies-4 wavelet coefficients are: $b_0 = c_3$, $b_1 = -c_2$, $b_2 = c_1$, and $b_3 = -c_0$. Note, that the Daubechies-4 wavelet has a support length of 3 and approximate continuous signals more accurately than do Haar wavelet (Kaplan, 2002). In the implementation provided the Daubechies-4 wavelet coefficients are: $b_0 = -c_3$, $b_1 = c_2$, $b_2 = 1c_1$, and $b_3 = c_0$ which accounts for shifting in the subbands LL, LH, HL and HH.

Case 3. m=3, Lenth-6 Daubechies Scaling Function. We apply the same conditions as in case 2 to yield the following system of equations:

$$c_0^2 + c_1^2 + c_2^2 + c_3^2 + c_4^2 + c_5^2 = 1 \tag{50}$$

$$c_0 + c_1 + c_2 + c_3 + c_4 + c_5 = \sqrt{2} \tag{51}$$

$$b_0 + b_1 + b_2 + b_3 + b_4 + b_5 = 0 \tag{52}$$

$$0b_0 + 1b_1 + 2b_2 + 3b_3 + 4b_4 + 5b_5 = 0 \tag{53}$$

$$0b_0 + 1^2b_1 + 2^2b_2 + 3^2b_3 + 4^2b_4 + 5^2b_5 = 0 \tag{54}$$

Solving this system of equations, we obtain Daubechies-6 scaling coefficients:

$$c_0 = \frac{1+\sqrt{10}+3\sqrt{5+2\sqrt{10}}}{16} = 0.47046721$$

$$c_1 = \frac{5+\sqrt{10}+\sqrt{5+2\sqrt{10}}}{16} = 1.14111692$$

$$c_2 = \frac{10-2\sqrt{10}+2\sqrt{5+2\sqrt{10}}}{16} = 0.650365$$

$$c_3 = \frac{10-2\sqrt{10}-2\sqrt{5+2\sqrt{10}}}{16} = -0.19093442$$

$$c_4 = \frac{5+\sqrt{10}-3\sqrt{5+2\sqrt{10}}}{16} = -0.12083221$$

$$c_5 = \frac{1+\sqrt{10}-\sqrt{5+2\sqrt{10}}}{16} = 0.0498175$$

Daubechies-6 wavelet coefficients $b_0 = c_5$, $b_1 = -c_4$, $b_2 = c_3$, $b_3 = -c_2$, $b_4 = c_1$ and $b_5 = -c_0$. The Daubechies-4 wavelet has a support length of 5.

The Daubechies-2m family has $2^{[m/4]-1}$ different solutions. For m=1,2,3 there is only one set of scaling functions and associated wavelets. Table 1 lists the scaling coefficients for m = 1,. . . , 8, where for $m > 3$ we choose the solution with most nearly linear phase. As the number of coefficients increases, the wavelet become more rounded and the associated band-pass spectrum more compact.

Using the approach described by M.J.T. Smith and T.P. Barnwell (1986) for obtaining exact-reconstruction filter banks, R. Ansari and all present conjugate-quadrature and linear-phase solutions for two-channel filter banks using Lagrange halfband filters. It is shown that the wavelet solutions obtained by I. Daubechies (1988) under certain regularity conditions are the same as the conjugate-quadrature solutions derived from Lagrange halfband filters using the above approach. The linear-phase solution that is described provides filters with simple coefficients. (R. Ansari, C. Guillemot, J. F. Kaiser, "Wavelet construction using Lagrange halfband filters", *IEEE Trans. Circuits Syst.*, vol.38, pp.1116-1118, Sept. 1991). See also *Zhuoer Shi; Wei, G.W.; Kouri, D.J.; Hoffman, D.K.; Zheng Bao;* "Lagrange wavelets for signal processing" Image Processing, IEEE Transactions, Volume: 10 Issue: 10 , Oct 2001 Page(s): 1488 -1508

6.4 Construction of the Daubechies Scaling and Wavelet Functions

The Daubechies wavelets and scaling functions have no analytical description. There are many approaches for displaying the graph of the Daubechies scaling functions, wavelets, and their Fourier transforms. We present two algorithms: one for constructing the scaling function and wavelet in the time domain and another algorithm for constructing the scaling function in the frequency domain. Both iterative algorithms are based on the scaling equation

$$\phi(x) = \sqrt{2} \sum_{k=0}^{N-1} c_k \phi(2x - k) \tag{55}$$

The associated wavelets and their Fourier transforms may be constructed by applying the same procedure to (55).

6.5 Detail Construction Algorithm

The algorithm used to construct the scaling and wavelet functions here was based on the scaling function equation. The general algorithm used is a recursive method of calculating each point into the routine.
For each value "x" passes in the algorithm

1. Checks to see if the value is greater or equal to 3 or less than or equal to 0, is so it returns zero
2. Checks to see is the value is equal 1 or if so it returns the appropriate $\varphi(1)$
3. Checks to see if the value is equal 2 or if so it returns the appropriate $\varphi(2)$
4. If none of the above are true then it recursively calculates $\varphi(2x)$ by calling itself with the value $2x$
5. Once that returns it multiplies the return value by c_0
6. It then recursively calculates $\varphi(2x - 1)$ by calling itself with the value $2x$
7. Once that returns it multiplies the return value by c_1
8. It then recursively calculates $\varphi(2x - 2)$ by calling itself with the value $2x$
9. Once that returns it multiplies the return value by c_2

Algorithm 6 Constructing the scaling function and the wavelet in the time domain

The algorithm is based on the following iterations

$$\phi^{(n+1)}(x) = \sqrt{2}\sum_{k=0}^{2} c_k \phi^{(n)}(2x-k) \qquad n = 0, 1, 2, ... \tag{56}$$

where

$$\phi^{0}(x) = \begin{cases} 1 & if \quad 0 \le x \le 1 \\ 0 & otherwice \end{cases} \tag{57}$$

from which we may compute all values of x.

Example Let $m = 2$, general steps:

Step 1. Upsample $\phi^{(0)}(x)$ from x to 2x
Step 2. Convolve (c_0, c_1) with the upsampled sequence to obtain $\phi^{(1)}(n)$

$$\phi^{(1)}(n) = c_0\phi^{(0)}(2n) + c_1\phi^{(0)}(2n-1) \tag{58}$$

Step 2. Generate

$$\phi^{(2)}(n) = c_0\phi^{(1)}(2n) + c_1\phi^{(1)}(2n-1) \tag{59}$$

by upsampling the sequence $\phi^{(1)}(n)$, then convolving it with (c_0, c_1).....
Step m. Generate

$$\phi^{(m)}(n) = c_0\phi^{(m-1)}(2n) + c_1\phi^{(m-1)}(2n-1) \tag{60}$$

by iterative upsampling and convolution.

By repeating this procedure we obtain the discretized $\phi(x)$ to an arbitrarily resolution. Note that the limit $\phi(n) = Lim_{n\to\infty}\phi^{(n)}(n)$ would satisfy the scaling equation (55).

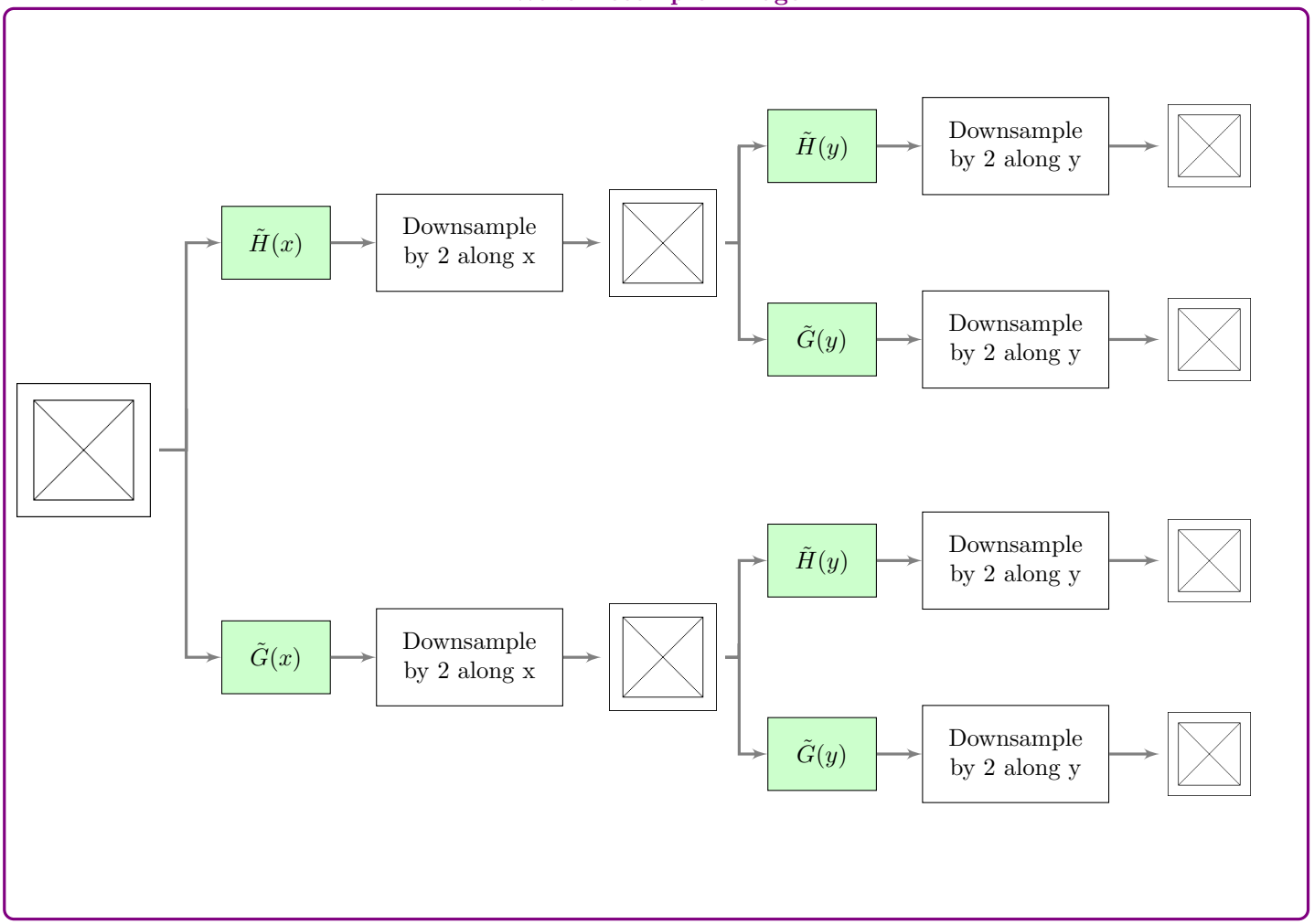


Figure 13: Wavelet Decomposition of Image

10. It then recursively calculates $\varphi(2x-3)$ by calling itself with the value $2x$
11. Once that returns it multiplies the return value by c_3
12. It then sums all the values and returns

In practical applications, a few iteration steps, typically fifty, are enough to obtain the scaling and wavelet functions. Initial values:

$$\varphi(x) = 0 \qquad x \leq 0, x \geq 3, \qquad \varphi(1) = \frac{1+\sqrt{3}}{2}, \qquad \varphi(2) = \frac{1-\sqrt{3}}{2} \tag{61}$$

The following Figure 13 display the iterative construction of the Daubechies-4 scaling and wavelet functions.

Daubechies Wavelet Family	***Coefficients***			
D_2	c_0	1	c_1	1
D_4	c_0	0.6830	c_2	0.3170
	c_1	1.1830	c_3	-0.1830
D_6	c_0	0.4705	c_3	-0.1909
	c_1	1.1411	c_4	-0.1208
	c_2	0.6504	c_5	0.0498
D_8	c_0	0.3258	c_4	-0.2645
	c_1	1.0109	c_5	0.0436
	c_2	0.8922	c_6	0.0465
	c_3	-0.0396	c_7	-0.0150
D_{10}	c_0	0.2264	c_5	-0.0456
	c_1	0.8539	c_6	0.1097
	c_2	1.0243	c_7	-0.0088
	c_3	0.1958	c_8	-0.0178
	c_4	-0.3427	c_9	0.0047
D_{12}	c_0	0.1577	c_6	0.1379
	c_1	0.6995	c_7	0.0389
	c_2	1.0623	c_8	-0.0447
	c_3	0.4458	c_9	0.0008
	c_4	-0.3200	c_{10}	0.0068
	c_5	-0.1835	c_{11}	-0.0015
D_{14}	c_0	0.1101	c_7	0.1140
	c_1	0.5608	c_8	-0.0538
	c_2	1.0311	c_9	-0.0234
	c_3	0.6644	c_{10}	0.0177
	c_4	-0.2035	c_{11}	0.0006
	c_5	-0.3168	c_{12}	-0.0025
	c_6	0.1008	c_{13}	0.0005
D_{16}	c_0	0.0770	c_8	-0.0246
	c_1	0.4425	c_9	-0.0624
	c_2	0.9555	c_{10}	0.0198
	c_3	0.8278	c_{11}	0.0124
	c_4	-0.0224	c_{12}	-0.0069
	c_5	-0.4017	c_{13}	-0.0006
	c_6	0.0007	c_{14}	0.0010
	c_7	0.1821	c_{15}	-0.0002
D_{18}	c_0	0.0539	c_9	-0.0956
	c_1	0.3448	c_{10}	0.0004
	c_2	0.8553	c_{11}	0.0316
	c_3	0.9295	c_{12}	-0.0067
	c_4	0.1884	c_{13}	-0.0061
	c_5	-0.4148	c_{14}	0.0026
	c_6	-0.1370	c_{15}	0.0003
	c_7	0.2101	c_{16}	-0.0004
	c_8	0.0435	c_{17}	-0.00006

Table 1: Daubechies scaling function coefficients

7 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) is the most widely used application in the fields of science and engineering, such as mathematics (linear systems, random process, probability, boundary-value systems), physics (quantum mechanics, optics, acoustics, astronomy), chemistry (spectroscopy, crystallography), and engineering (digital signal and image processing, telecommunications, computer vision). The theory of trigonometric series can be dated back to the beginning of 18th century. In 1747, Euler represented the movements of the planets in the form of a trigonometric series, which actually contained what is now called the Fourier series (Euler, 1760). In 1754, d'Alambert represented the reciprocal value of the mutual distance of two planets as a series of cosine functions (d'Alambert, 1754). H. H. Goldstine attributed to Carl Friedrich Gauss an algorithm, developed in 1805, similar to the fast Fourier transform (FFT) for the computation of the coefficients of a finite Fourier series (Goldstine, 1977). In 1807, Fourier discovered that a wide class of signals could be generated by summing scaled sine and cosine functions. These early techniques provided ideal tools for analyzing both periodic signals and the more general class of stationary signals. In 21st century, discrete Fourier transform remains one of the most frequently applied tools in science and technology. This is also because of the development in fast Fourier transform algorithms. The history of FFT development, dates back to Gauss, can be found in ((Goldstine, 1977; Caglayan, 2008).

Consider the N-point discrete Fourier Transform (DFT) of a signal $x(n), n = 0, 1, 2, \cdot, N-1$ is defined by

$$Y(k) = \sum_{n=0}^{N-1} x(n)w(n,k), \qquad k = 0, 1, \cdot, N-1 \tag{62}$$

Similarly, the inverse Fourier transform of the sequence $x(n) = (x_1, x_2, \cdot, x_{N-1}), n = 0, 1, 2, \cdot, N-1$ can be given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y(k)w^{-1}(n,k), \qquad n = 0, 1, \cdot, N-1 \tag{63}$$

where $w(n,k)$ are the complex roots of unity, that is

$$w(n,k) = e^{-2i\pi k/N}, \qquad n, k = 0, 1, \cdot, N-1 \tag{64}$$

Both equations (62) and (63) can be expressed in vector matrix form as

$$Y_N = \mathbf{F_N} x_N \tag{65}$$

$$x_N = \frac{1}{N} \mathbf{F_N^{-1}} Y_N \tag{66}$$

DFT matrix of size $N \times N$ composed of the twiddle factors as:

$$\mathbf{F_N} = \begin{bmatrix} w(0,0) & w(0,1) & w(0,2) & \cdots & w(0,N-1) \\ w(1,0) & w(1,1) & w(1,2) & \cdots & w(1,N-1) \\ w(1,0) & w(1,1) & w(1,2) & \vdots & w(2,N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w(N-1,0) & \cdots & \cdots & \cdots & w(N-1,N-1) \end{bmatrix} \tag{67}$$

Figure 14 illustrates the basis function of the discrete Fourier transform when $N = 16$, in which DFT matrix coefficients in each row is represented by a linear combination of each element of Fourier space, i.e. sines and cosines.

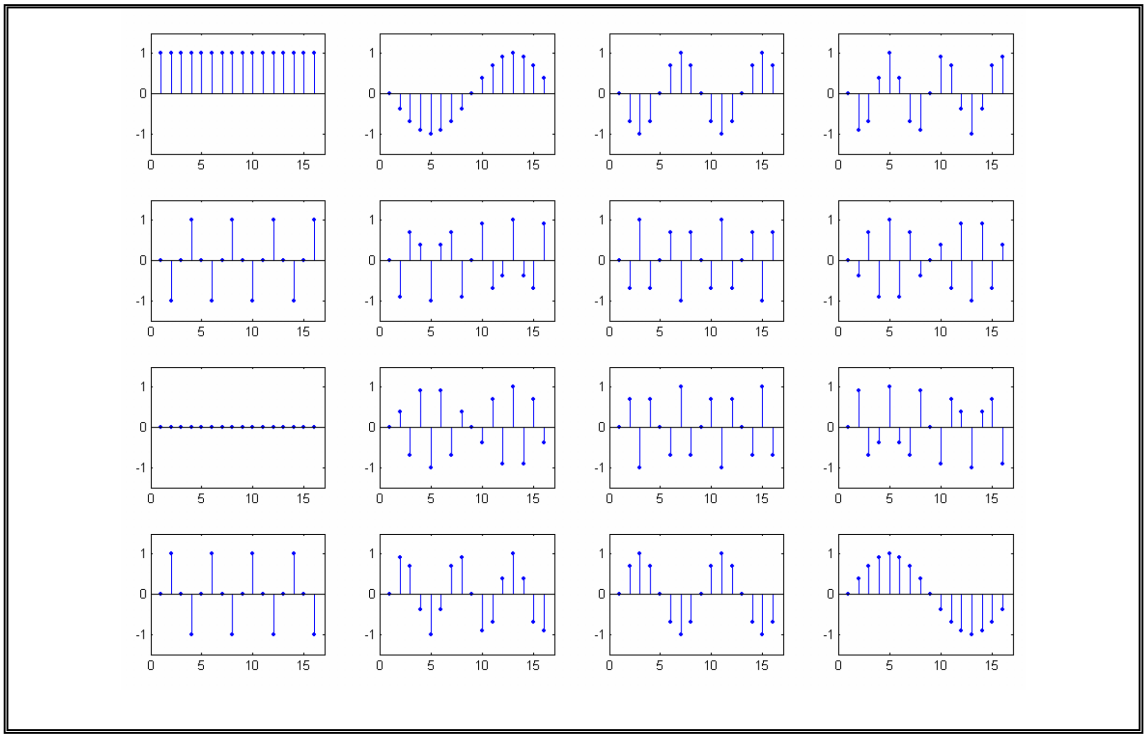


Figure 14: 16-point basis functions of discrete Fourier transform

The DFT is used in signal processing to identify frequencies of interest. The identification of frequencies allows the use of digital filters to eliminate noise and audio components outside of the frequency of interest. For example if human vocals are needed to be extracted a bandpass filter can be used to extract vocal components of audio file around 8k Hz. The DFT has been extended to 2-dimensional files such as images. The frequencies extracted using the 2-dimensional DFT used with images can smooth an image by removing high frequencies in an image. Caglayan in 2008 showed his work using the DFT in extracting features from a large volume of images for the use of determining images that have been modified from their original form.

7.1 1D Discrete Fourier Transform (DFT) Algorithm

Below is a code snippet that outlines the basic algorithm for computing the DFT of a one-dimensional signal. The function DFT takes a one-dimensional array $\mathbf{x}$ of length N as its input and returns the Fourier-transformed array Y.

In this LaTeX example, we loop through each output element $\mathbf{x}$ and each input element $x(n), n = 0, 1, \cdots, N-1$. The twiddle factor w is calculated and used to accumulate into $x(n)$. Finally, the Fourier-transformed array Y is returned.

This is a straightforward, $O(N^2)$ complexity example. In practice, faster algorithms like the Fast Fourier Transform (FFT) are often used to compute the DFT more efficiently.

Algorithm 7 Discrete Fourier Transform (DFT)

```
function DFT(x)
    N ← length(x)                                   ▷ Get the length of the input signal
    Initialize complex array Y[0...N − 1] with zeros ▷ Initialize output array
    for k = 0 → N − 1 do                            ▷ Loop through each output element
        for n = 0 → N − 1 do                        ▷ Loop through each input element
            w ← exp(−j (2π/N) kn)                   ▷ Compute twiddle factor
            Y[k] ← Y[k] + x[n] × w                  ▷ Accumulate result
        end for
    end for
    return Y                                        ▷ Return Fourier-transformed array
end function
```

7.2 2D Discrete Fourier Transform (DFT)

The 2D Discrete Fourier Transform (2D-DFT) is an extension of the 1D Discrete Fourier Transform and is commonly used in image and signal processing. The 2D-DFT transforms a 2D array (often representing an image) into its frequency components, capturing both phase and magnitude information. The resulting 2D array can be used for various applications like image filtering, compression, and other types of analysis.

Mathematical Formulation

The 2D-DFT of a 2D array $Im(n_1, n_2)$ of dimensions $N_1 \times N_2$ is given by:

$$Y(k_1, k_2) = \sum_{n_1=0}^{N-1-1} \sum_{n_2=0}^{N_2-1} Im(n_1, n_2) \exp\left(-j\frac{2\pi}{N_1}k_1 n_1 - j\frac{2\pi}{N_2}k_2 n_2\right) \tag{68}$$

Here, $Y(k_1, k_2)$ are the Fourier coefficients corresponding to the frequencies k_1 and k_2, $Im(n_1, n_2)$ are the pixel values in the spatial domain, and j is the imaginary unit $\sqrt{-1}$.

Inverse 2D-DFT

The inverse transform can be used to recover $Im(n_1, n_2)$ from $Y(k_1, k_2)$ and is given by:

$$Im(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y(k_1, k_2) \exp\left(j\frac{2\pi}{N_1}k_1 n_1 + j\frac{2\pi}{N_2}k_2 n_2\right) \tag{69}$$

Algorithmic Complexity

The direct computation of the 2D-DFT has a time complexity of $O(N_1^2 N - 2^2)$, which can be quite slow for large images. However, efficient algorithms, like the 2D Fast Fourier Transform (FFT), reduce this to $O(N_1 N_2 \log(N_1 N_2))$

The 2D-DFT is a powerful tool for analyzing and processing images and other 2D signals. It is a cornerstone in fields like computer vision, medical imaging, and telecommunications.

7.2.1 2D Discrete Fourier Transform (DFT) Algorithm

The 2D Discrete Fourier Transform (2D-DFT) is an extension of the 1D Discrete Fourier Transform (DFT) and is commonly used in image and signal processing. Below is a description outlining the 2D-DFT algorithm. The function TwoDDFT takes a 2D array Im of dimensions $N_1 \times N_2$ as input and returns the 2D-DFT-transformed array Y.

Algorithm 8 2D Discrete Fourier Transform (2D-DFT)

function TwoDDFT(Im)
  $N_1, N_2 \leftarrow$ dimensions of Im ▷ Get dimensions of the 2D array
  Initialize complex array $Y[0 \dots N_1 - 1][0 \dots N_2 - 1]$ with zeros ▷ Initialize output array
  for $k_1 = 0 \rightarrow N_1 - 1$ **do** ▷ Loop through each output row
    for $k_2 = 0 \rightarrow N_2 - 1$ **do** ▷ Loop through each output column
      for $n_1 = 0 \rightarrow N_1 - 1$ **do** ▷ Loop through each input row
        for $n_2 = 0 \rightarrow N_2 - 1$ **do** ▷ Loop through each input column
          $w \leftarrow \exp\left(-j\frac{2\pi}{N_1}k_1 n_1 - j\frac{2\pi}{N_2}k_2 n_2\right)$ ▷ Compute twiddle factor
          $Y[k_1][k_2] \leftarrow Y[k_1][k_2] + Im[n_1][n_2] \times w$ ▷ Accumulate result
        end for
      end for
    end for
  end for
  return Y ▷ Return 2D-DFT-transformed array
end function

In this algorithm, we loop through each output element $Y[k_1][k_2]$ and each input element $Im[n_1][n_2]$. The twiddle factor w is calculated and used to accumulate into $Y[k_1][k_2]$. Finally, the 2D-DFT-transformed array Y is returned.

This is a straightforward $O(N_1^2 N_2^2)$ complexity example. In practice, faster algorithms like the 2D Fast Fourier Transform (FFT) are often used to compute the 2D-DFT more efficiently.

8 Discrete Cosine Transform (DCT)

The standard DCT used in JPEG compression has two properties, i.e., the directional and frequency distributions of 8×8 blocks within an image (Rao and Yip, 1990). In JPEG compression on a two dimensional (2-D) signal, the zig-zag scan shown in Figure 15(a) is used to take advantage of the frequency distributions of the DCT shown in Figure 15(b) (Brown and Shepherd, 1995, pp. 224). The DCT decomposition divides the coefficients into low, medium and high frequencies. Figure 15(c) shows the breakdown of the vertical, diagonal and horizontal directions of the coefficients. In this research both the frequencies and directions of the DCT are investigated to generate features. Figure 15(d) shows an 8×8 image with a horizontal edge between black and white pixels. The corresponding 2-D DCT of Figure 15(d) is shown in Figure 15(g) which has coefficients that are prominent along the first column. In Figure 15(e) an image is shown with a diagonal edge between black and white pixels with a corresponding 2-D DCT shown in Figure 15(h) which has coefficients located along the diagonal. In Figure 15(f) an image is shown with a vertical edge between black and white pixels with a corresponding 2-D DCT shown in Figure 15(i) which has coefficients located along the first row.

8.1 JPEG Image Representation Background

In this section, the basic structure of the JPEG image format and the steps in the compression process are described. This is followed by a brief introduction of JPEG image embedding methods.

The Joint Photographic Experts Group (JPEfG) format uses lossy compression to achieve high levels of compression on images with many colors (Elysium Ltd., 2004). JPEG is an international standard for still image compression, and is widely used for compressing gray scale and color images. JPEG images are commonly used for storing digital photos, and publishing Web graphics; tasks for which slight reductions in the image quality are barely noticeable. Due to the loss of quality during the compression process, JPEGs should be used only where image file size is important (Murry and vanRyper, 1994; Brown and Shepherd, 1995).

The JPEG encoder, shown in figure below, performs compression with the following sequential steps: image preprocessing (divides the input image into 8×8 blocks), forward DCT of each 8×8 block, quantization with scaling factor, separation of DC and AC coefficients, prediction of the DC coefficient and zig-zag scan the AC coefficients and Huffman encoder (there is a separate encoder for the DC and AC coefficients).

In JPEG decoding, all steps from the encoding process are reversed. The following procedure is a short description of the JPEG baseline systems.

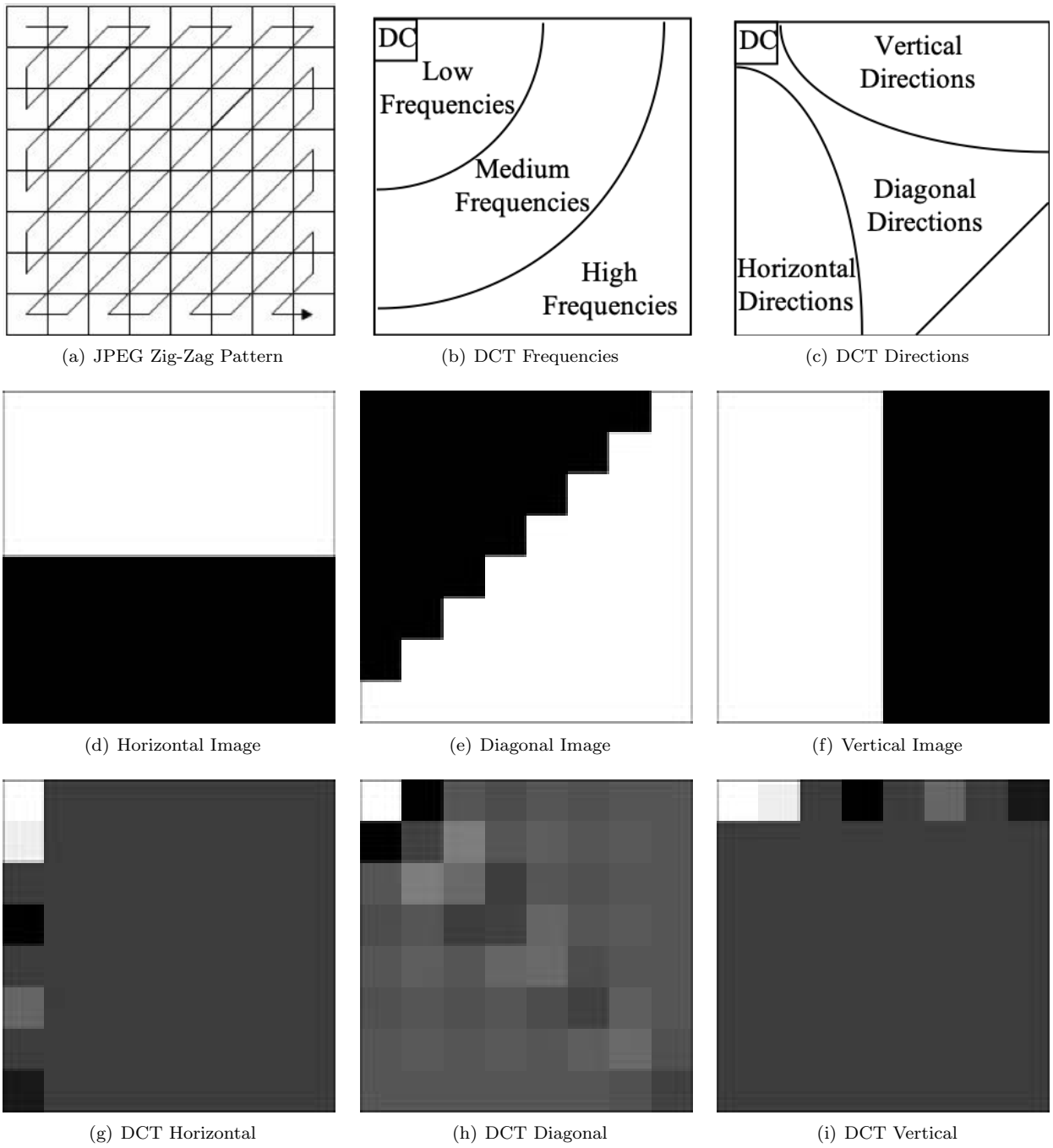
Preprocessing block - Subdivides the image into blocks of 8×8 pixels and level-shift the original pixel values from the range $[0, 225]$ to the range $[-128, +127]$ by subtracting 128. The shifting procedure is a preprocessing step for the DCT calculation.

Forward DCT block - Perform a two dimensional discrete cosine transform (DCT) on each level-shifted block B from the Preprocessing block step. The two dimension DCT is defined as

$$C(n_1, n_2, k_1, k_2) = \begin{cases} \sqrt{\frac{1}{N_1 N_2}} & \text{if } k_1 = k_2 = 0 \\ \sqrt{\frac{2}{N_1 N_2}} \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right) & \text{if } 1 \leq k_1 \leq N_1 - 1 \text{ and } 1 \leq k_2 \leq N_2 - 1 \end{cases} \tag{70}$$

where $0 \leq n_1 \leq N_1 - 1$ and $0 \leq n_2 \leq N_2 - 1$. Assuming that $N_2 = 1$, will give $k_2 = n_2 = 0$ or simply non existent. This will result in the following updated equation

$$C(n, k) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } k = 0 \\ \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right)\right) & \text{if } 0 \leq n \leq N - 1 \text{ and } 1 \leq k \leq N - 1 \end{cases} \tag{71}$$



(a) JPEG Zig-Zag Pattern (b) DCT Frequencies (c) DCT Directions

(d) Horizontal Image (e) Diagonal Image (f) Vertical Image

(g) DCT Horizontal (h) DCT Diagonal (i) DCT Vertical

Figure 15: DCT decomposition (a) zig-zag scan pattern (b) low, medium and high frequency distributions (c) vertical, diagonal and horizontal directions (d) 8×8 image with a horizontal edge between pixels (e) 8×8 image with a diagonal edge between pixels (f) 8×8 image with a vertical edge between pixels (g) 2-D DCT representation of horizontal image (h) 2-D DCT representation of diagonal image (i) 2-D DCT representation of vertical image.

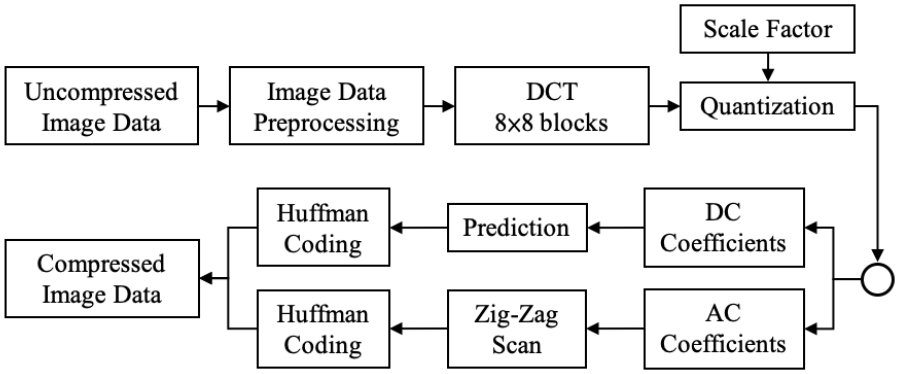


Figure 16: Block Diagrams of Sequential JPEG Encoder.

This provides an $N \times N$ matrix for use to transform an $N \times N$ input matrix. The transform is performed on the two dimensional matrix Im as $CImC^T$. In the case when matrix multiplication is not used the following equation can be used to take the two-dimensional discrete cosine transform of the matrix Im

$$CImC^T = Y_{DCT} = d_{k_1} d_{k_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} Im_{k_1,k_2} \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right) \tag{72}$$

$$\text{for } 1 \leq k_1 \leq N_1 - 1 \text{ and } 1 \leq k_2 \leq N_2 - 1$$

where

$$d_{k_1} = \begin{cases} \sqrt{\frac{1}{N_2}} & \text{if } k_1 = 0 \\ \sqrt{\frac{2}{N_1}} & \text{if } 1 \leq k_1 \leq N_1 - 1 \end{cases} \tag{73}$$

and

$$d_{k_2} = \begin{cases} \sqrt{\frac{1}{N_2}} & \text{if } k_2 = 0 \\ \sqrt{\frac{2}{N_2}} & \text{if } 1 \leq k_2 \leq N_2 - 1 \end{cases} \tag{74}$$

The transform helps to remove data redundancy by mapping data from a spatial domain to the discrete cosine domain. No compression has been achieved in this stage, but by changing representation of the information contained in the image block it makes the data more suitable for compression. In the JPEG encoding process, $N_1 = N_2 = 8$ which requires an 8-point DCT as shown in Equation 75. The values in Equation 75 are calculated using Equation 71 where $N = 8$

$$C(n, k_1) = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \tag{75}$$

Quantization - Quantize the DCT coefficients block obtained from the previous step using the quantization table Q. The quantization table is a matrix used to divide the transformed block for compression purpose by reducing the amplitude of the DCT coefficient values and increasing the number of zero valued coefficients. The Huffman encoder takes advantage of these quantized values. When Qs is represented the value s is a scalar multiple, called the scale (or quality) factor, which defines the amount of compression within the image. Higher values of s yield higher compression. The below figure shows an instance of the typical quantization matrix Qs.

$$Qs = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 6 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{76}$$

A set of four quantization tables are specified by the JPEG standard (Independent JPEG Group, 1998). After quantization, most of the DCT coefficients in the 8×8 blocks are truncated to zero values. It is the principal of lossiness in the JPEG transform-based encoder.

DC Coefficient Coding - The first coefficient, coefficient 1 (upper left) in figure b) below is called the "DC coefficient", short for the direct current coefficient, and represents the average brightness (intensity) of the component block. To encode the DC coefficient, the JPEG standard utilizes a Huffman difference code table that categorizes the value according to the number of k bits that are required to represent its magnitude. The value of the element is encoded with k bits.

AC Coefficients Coding - The remaining 63 coefficients are the "AC coefficients", short for the alternating current coefficients. The Huffman code assigns short (binary) codewords to each AC coefficient. The AC coefficient encoding scheme is slightly more elaborate than the one for the DC coefficient. For each AC array, a run-length of 0 elements is recorded. When encountering a non-zero element, the length of $0s$ is recorded and the number of k bits to represent the magnitude of the element is determined. The run-length and k bits are used as a category in the JPEG default Huffman table for assigning a code.

Using a zig-zag run encoder converts the 8×8 array of DCT coefficients into a column vector of length k (zig-zag goes from left to right and top to bottom). The "zig-zag" scan attempts to trace the DCT coefficients according to their significance, shown in figure below.

The coefficient ordering sequence is shown below.

$$\begin{bmatrix} 1 & 2 & 6 & 7 & 15 & 16 & 28 & 29 \\ 3 & 5 & 8 & 14 & 17 & 27 & 30 & 43 \\ 4 & 9 & 13 & 18 & 26 & 31 & 42 & 44 \\ 10 & 12 & 19 & 25 & 32 & 41 & 45 & 54 \\ 11 & 20 & 24 & 33 & 40 & 46 & 53 & 55 \\ 21 & 23 & 34 & 39 & 47 & 52 & 56 & 61 \\ 22 & 35 & 38 & 48 & 51 & 57 & 60 & 62 \\ 36 & 37 & 49 & 50 & 58 & 59 & 63 & 64 \end{bmatrix} \tag{77}$$

The Huffman encoding reduces the number of bits needed to store each of the 64 integer coefficients. For example, when a true color uncompressed image of size 512×512 pixels is stored the file size is 769 kilobytes. However, this same image store as a JPEG at a quality factor of 75, the image is stored in 200 kilobytes or smaller. The Huffman encoding tables for the DC and AC coefficients can be found in Gonzalez and Woods (2007), Elysium Ltd. (2004), Independent JPEG Group (1998), and JPEG (1994).

One of the primary reasons using image embedding methods for creating stego files is due to the number of redundant portions within a digital image. The vast number of JPEG images on the Internet makes them ideal cover images for hiding secrete data and transmitting them as stego images. In JPEG steganography, the stego message is converted to binary values and embedded into DC and AC coefficients prior to Huffman encoding. By embedding at this stage, the stego message can be extracted without losing the message. The embedding methods range from simple embedding techniques that alter the least significant bits (LSB) of the coefficients such as JP Hide (Latham, 1999) and JSteg (Upham, 1993) to more complicated embedding techniques that maintain natural histograms of the coefficients such as; F5 (Westfeld, 2001; 2003), JP Hide (Latham, 1999), JSteg (Upham, 1993), Model-base (Sallee, 2003; 2006), Model-based Version 1.2 (Sallee, 2008a), OutGuess (Provos, 2004), Steganos (2008), StegHide (Hetzl, 2003) and UTSA (Agaian et al., 2006). The six tools selected provide a set of embedding methods that differ in embedding strategy. Investigation of these methods has provided an insight into six different and unique embedding capacities, embedding patterns and the appearance of the individual feature spaces. Another reason for selecting these particular tools is in previous research and existing steganalysis tools, these 6 embedding methods have been used for analysis (Provos and Honeyman, 2003; Lyu and Farid, 2004; Kharrazi et al., 2005; Shi et al., 2005; Xuan et al., 2005; Fu et al., 2006; Pevny and Fridrich, 2007).

In summary, a useful property of JPEG is that the degree of lossiness can be varied by adjusting the quality factor s (scale of the quantization table), shown in Figure 2.3. The ease of file sharing with JPEG images and its popularity over the internet has made JPEG image format a desirable cover file for many stego methods. Each embedding method leaves a signature that can be identified by various statistical measures. The next section describes feature generations methods used to identify changes made to a JPEG image.

8.1.1 Discrete Cosine Transform (DCT) Algorithm

Below is Algorithm 9 that outlines the 2D-DCT algorithm. The function 2D-DCT takes a 2D matrix Im of dimensions $N_1 \times N_2$ as input and returns the 2D-DCT-transformed matrix Y_{DCT}.

Algorithm 9 2D Discrete Cosine Transform (2D-DCT)

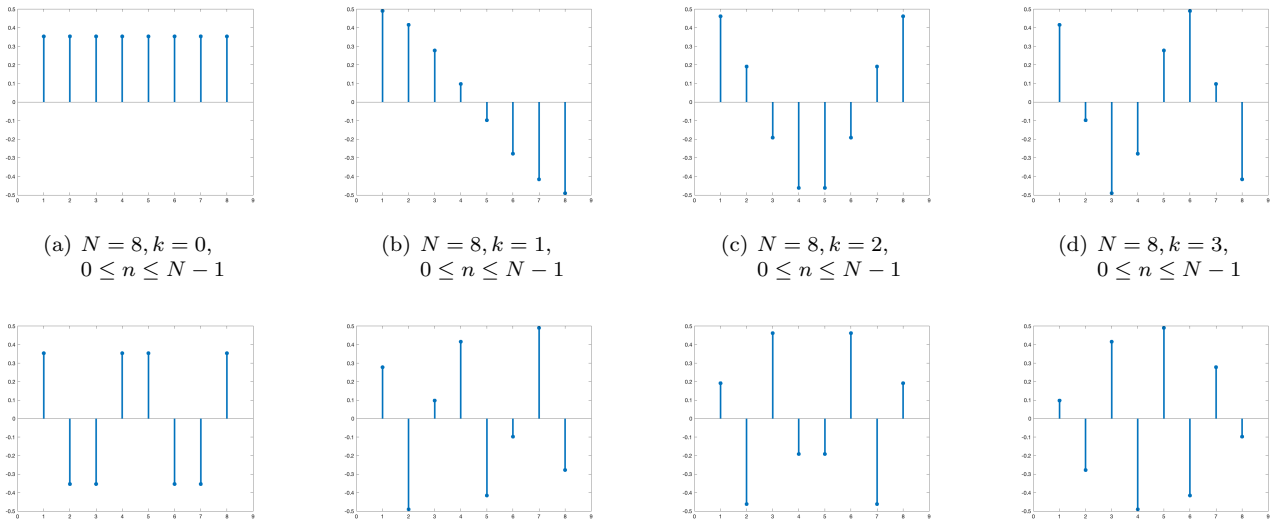
function 2D-DCT(Im)
  $N_1, N_2 \leftarrow$ dimensions of Im ▷ Get dimensions of the 2D matrix
  Initialize array $Y_{DCT}[0 \dots N_1 - 1][0 \dots N_2 - 1]$ with zeros ▷ Initialize output array
  for $k_1 = 0 \rightarrow N_1 - 1$ **do** ▷ Loop through each output row
    for $k_2 = 0 \rightarrow N_2 - 1$ **do** ▷ Loop through each output column
      for $n_1 = 0 \rightarrow N_1 - 1$ **do** ▷ Loop through each input row
        for $n_2 = 0 \rightarrow N_2 - 1$ **do** ▷ Loop through each input column
          $d_{k_1} \leftarrow$ if $(k_1 = 0)$ then $\sqrt{\frac{1}{N_1}}$ else $\sqrt{\frac{2}{N_1}}$
          $d_{k_2} \leftarrow$ if $(v = 0)$ then $\sqrt{\frac{1}{N_2}}$ else $\sqrt{\frac{2}{N_2}}$
          $Y_{DCT}[k_1][k_2] \leftarrow Y_{DCT}[k_1][k_2] + d_{k_1} \cdot d_{k2} \cdot Im_{k_1,k_2} \cdot \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cdot \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right)$
        end for
      end for
    end for
  end for
  return Y_{DCT} ▷ Return 2D-DCT-transformed array
end function

In Algorithm 9, we loop through each output element Y_{DCT} and each input element Im. The coefficients d_{k_1} and d_{k_2} are calculated to normalize the DCT, especially for the zero-frequency (DC) terms. The double summation operation calculates $Y_{DCT}[k_1][k_2]$ based on the input matrix Im and the DCT basis functions.

This is a simple and direct, $O(N_1^2 N_2^2)$ complexity example. In practice, faster algorithms are often employed to compute the 2D-DCT more efficiently.

8.2 Example - 8-Point DCT

In this example we consider a 8-point DCT. Evaluating the images in Figure 17 give a view of $N = 8, 0 \leq k \leq 7$, and $0 \leq n \leq 7$ where each image holds the value of k constant illustrates the basis function of the discrete Cosine transform. Each row in Equation 79 is represented by a linear combination of each element of discrete cosine space.



(a) $N = 8, k = 0$, $0 \leq n \leq N-1$ (b) $N = 8, k = 1$, $0 \leq n \leq N-1$ (c) $N = 8, k = 2$, $0 \leq n \leq N-1$ (d) $N = 8, k = 3$, $0 \leq n \leq N-1$

(e) $N = 8, k = 4$, $0 \leq n \leq N-1$ (f) $N = 8, k = 5$, $0 \leq n \leq N-1$ (g) $N = 8, k = 6$, $0 \leq n \leq N-1$ (h) $N = 8, k = 7$, $0 \leq n \leq N-1$

Figure 17: 8-Point Discrete Cosine Transform Coefficients

To generate each of the subfigures shown in Figure 17 the following example is provided and starts with Figure 18 which is equivalent to Figure 17 (a). The values in the figure are generated using Equation 71 and calculated letting $N = 8$, $k = 0$ and $0 \leq n \leq N-1$ resulting in Equation 71 being calculated as $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{8}} = 0.3536$.

Now considering Figure 17 (b) and the equivalent Figure 19 the equations will be calculated allowing $N = 8, k = 1$, and $0 \leq n \leq N-1$ resulting in Equation 71 being calculated as follows:

Let $N = 8$, $k = 1$ and $n = 0$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(0)+1)1}{2(8)}\right) = 0.4904$.

Let $N = 8$, $k = 1$ and $n = 1$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(1)+1)1}{2(8)}\right) = 0.4157$.

Let $N = 8$, $k = 1$ and $n = 2$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(2)+1)1}{2(8)}\right) = 0.2778$.

Let $N = 8$, $k = 1$ and $n = 3$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(3)+1)1}{2(8)}\right) = 0.0975$.

Let $N = 8$, $k = 1$ and $n = 4$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(4)+1)1}{2(8)}\right) = -0.0975$.

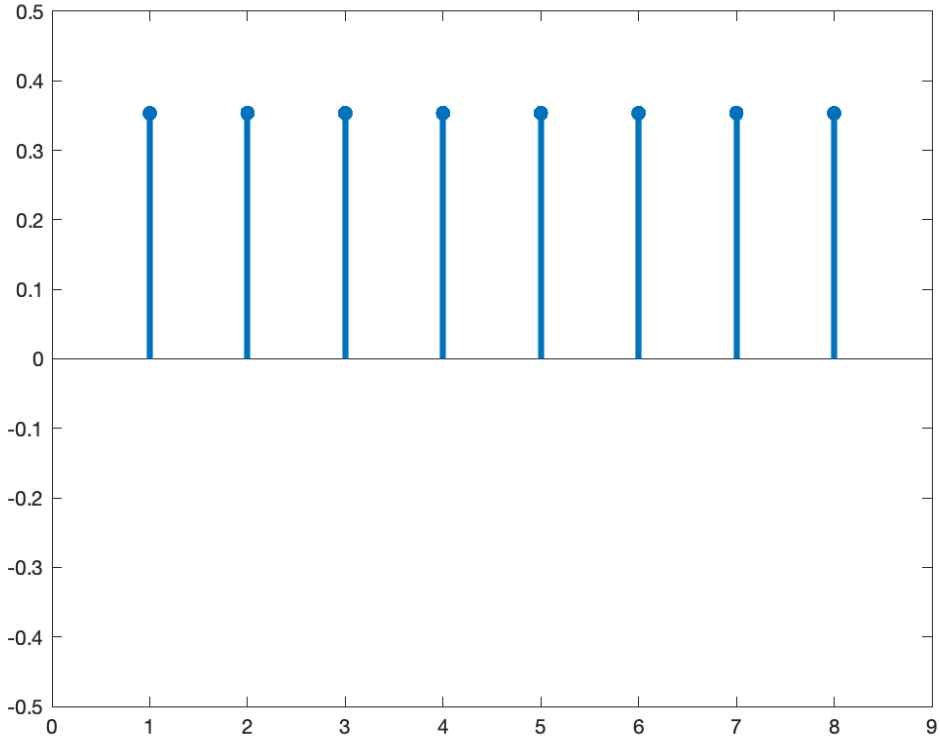


Figure 18: 8-point basis functions of discrete Cosine transform

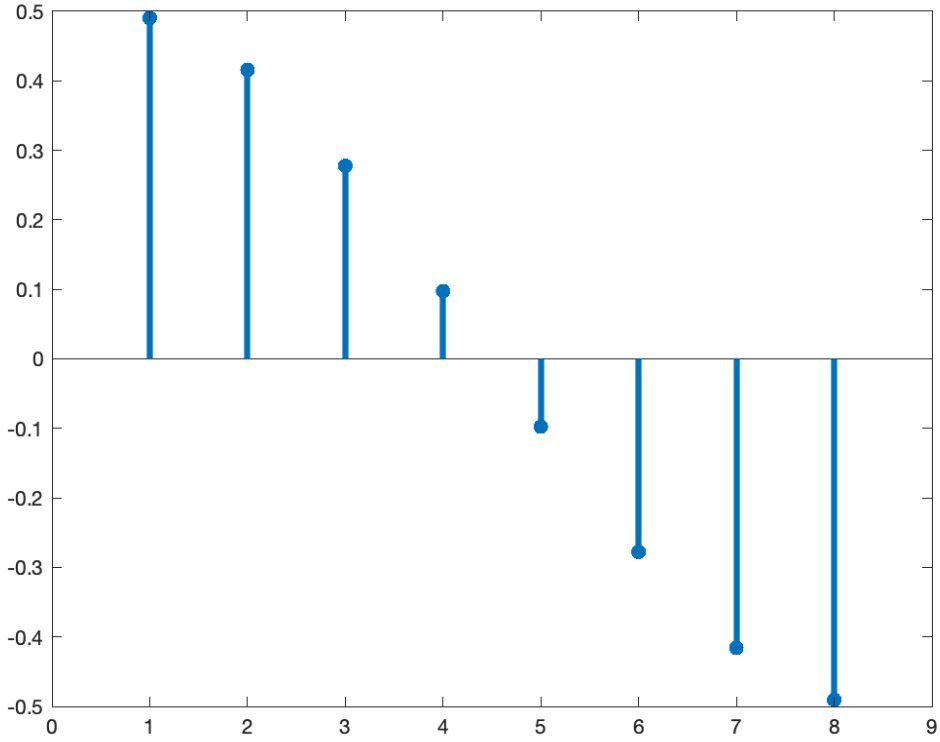


Figure 19: 8-point basis functions of discrete Cosine transform

Let $N = 8$, $k = 1$ and $n = 5$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(5)+1)1}{2(8)}\right) = -0.2778$.

Let $N = 8$, $k = 1$ and $n = 6$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(6)+1)1}{2(8)}\right) = -0.4157$.

Let $N = 8$, $k = 1$ and $n = 7$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(7)+1)1}{2(8)}\right) = -0.4904$.

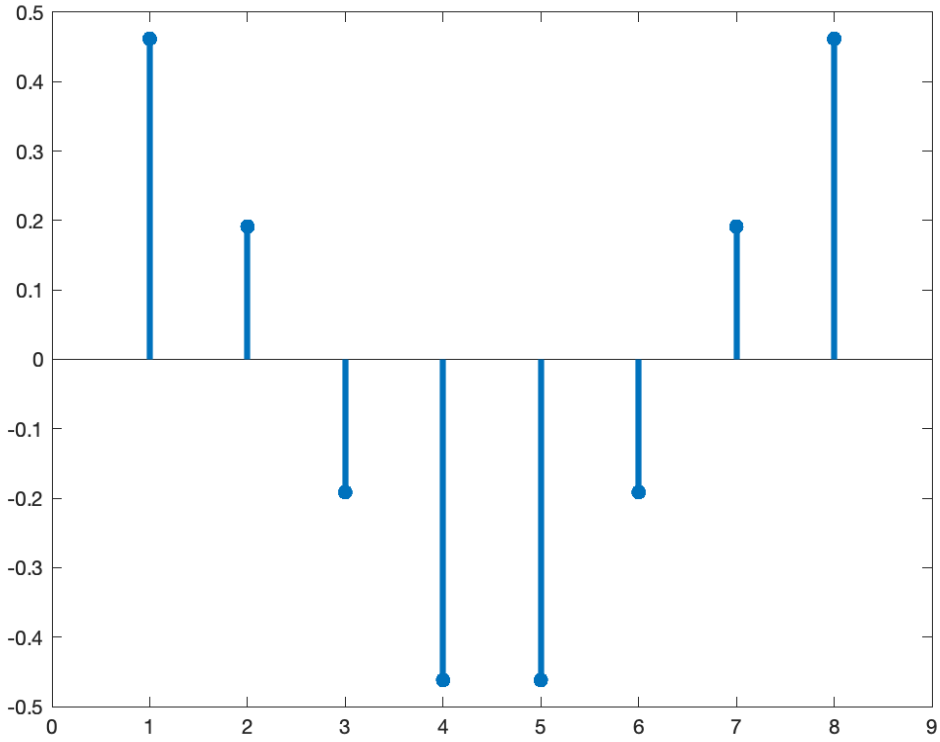


Figure 20: 8-point basis functions of discrete Cosine transform

Now considering Figure 17 (c) and the equivalent Figure 20 the equations will be calculated allowing $N = 8, k = 2$, and $0 \leq n \leq N - 1$ resulting in Equation 71 being calculated as follows:

Let $N = 8$, $k = 2$ and $n = 0$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(0)+1)2}{2(8)}\right) = 0.4619$.

Let $N = 8$, $k = 2$ and $n = 1$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(1)+1)2}{2(8)}\right) = 0.1913$.

Let $N = 8$, $k = 2$ and $n = 2$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(2)+1)2}{2(8)}\right) = -0.1913$.

Let $N = 8$, $k = 2$ and $n = 3$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(3)+1)2}{2(8)}\right) = -0.4619$.

Let $N = 8$, $k = 2$ and $n = 4$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(4)+1)2}{2(8)}\right) = -0.4619$.

Let $N = 8$, $k = 2$ and $n = 5$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(5)+1)2}{2(8)}\right) = -0.1913$.

Let $N = 8$, $k = 2$ and $n = 6$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(6)+1)2}{2(8)}\right) = 0.1913$.

Let $N = 8$, $k = 2$ and $n = 7$will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{8}} \cos\left(\frac{\pi(2(7)+1)2}{2(8)}\right) = 0.4619$.

Continuing this process will produce the results for the remaining subfigures in Figure 17. This process will result in Eq. 71 producing the 2-D DCT matrix as follows:

$$C(n,k) = C(n,k_1) \tag{78}$$

$$C(n,k_1) = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \tag{79}$$

8.2.1 Matlab Code

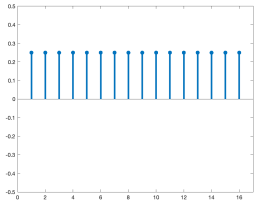
In order to create the 8-point DCT matrix, we used the Matlab code shown in Algorithm 10. The following code is based on Eq. 71 to generate an $N \times N$ N-point DCT matrix.

8.3 Example - 16-Point DCT

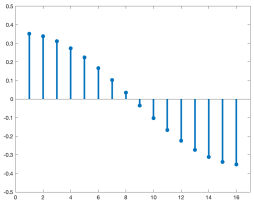
In this example we consider a 16-point DCT. Evaluating the images in Figure 21 give a view of $N = 16, 0 \leq k \leq 15$, and $0 \leq n \leq 15$ where each image holds the value of k constant illustrates the basis function of the discrete Cosine transform. Each row in Equations 81 and 82 are represented by a linear combination of each element of discrete cosine space.

To generate each of the subfigures shown in Figure 21 the following example is provided and starts with Figure 22 which is equivalent to Figure 21 (a). The values in the figure are generated using Equation 71 and calculated letting $N = 16$, $k = 0$ and $0 \leq n \leq N-1$ resulting in Equation 71 being calculated as $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{16}} = 0.2500$.

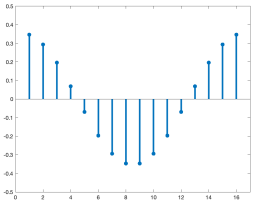
Now considering Figure 21 (b) and the equivalent Figure 23 the equations will be calculated allowing $N = 16, k = 1$, and $0 \leq n \leq N-1$ resulting in Equation 71 being calculated as follows:



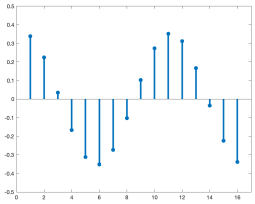
(a) $N = 16, k = 0$, $0 \leq n \leq N-1$



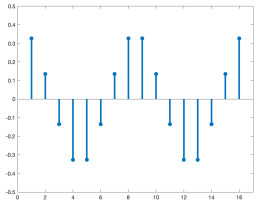
(b) $N = 16, k = 1$, $0 \leq n \leq N-1$



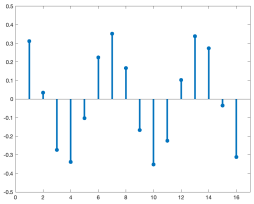
(c) $N = 16, k = 2$, $0 \leq n \leq N-1$



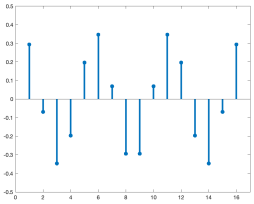
(d) $N = 16, k = 3$, $0 \leq n \leq N-1$



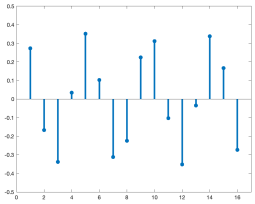
(e) $N = 16, k = 4$, $0 \leq n \leq N-1$



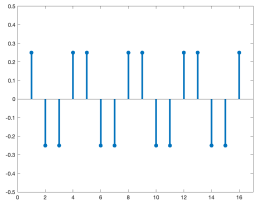
(f) $N = 16, k = 5$, $0 \leq n \leq N-1$



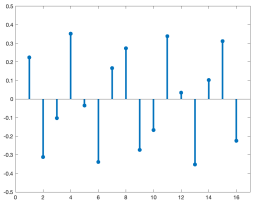
(g) $N = 16, k = 6$, $0 \leq n \leq N-1$



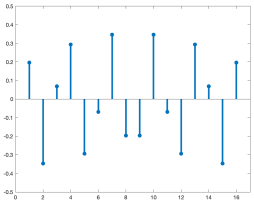
(h) $N = 16, k = 7$, $0 \leq n \leq N-1$



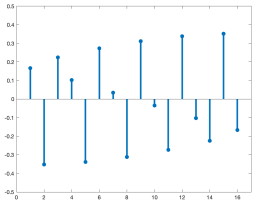
(i) $N = 16, k = 8$, $0 \leq n \leq N-1$



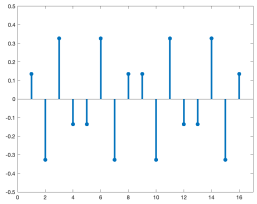
(j) $N = 16, k = 9$, $0 \leq n \leq N-1$



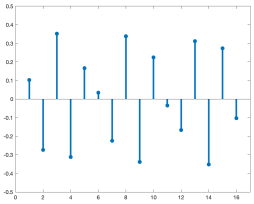
(k) $N = 16, k = 10$, $0 \leq n \leq N-1$



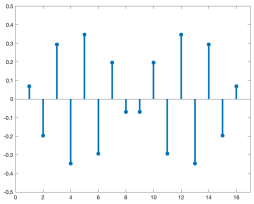
(l) $N = 16, k = 11$, $0 \leq n \leq N-1$



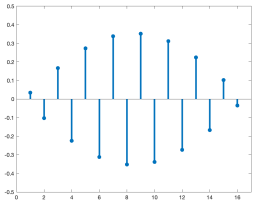
(m) $N = 16, k = 12$, $0 \leq n \leq N-1$



(n) $N = 16, k = 13$, $0 \leq n \leq N-1$



(o) $N = 16, k = 14$, $0 \leq n \leq N-1$



(p) $N = 16, k = 15$, $0 \leq n \leq N-1$

Figure 21: 16-Point Discrete Cosine Transform Coefficients

Algorithm 10 Matlab Code: N-point DCT

```
function NPDCT = NPointDCT2(N)

% This function produces the N-point DCT_2 of an input size received.
% The N-point DCT_2 matrix will be a square matrix in this function.

for k = 0:N-1
    for n = 0:N-1
        if k == 0
            NPDCT(k+1,n+1) = sqrt(inv(N));
        else
            temp = cos((pi*(2*n + 1)*k)/(2*N));
            NPDCT(k+1,n+1) = sqrt(2/N)*temp;%1/sqrt(N) = sqrt(2/N)
        end
    end
end
NPDCT;
```

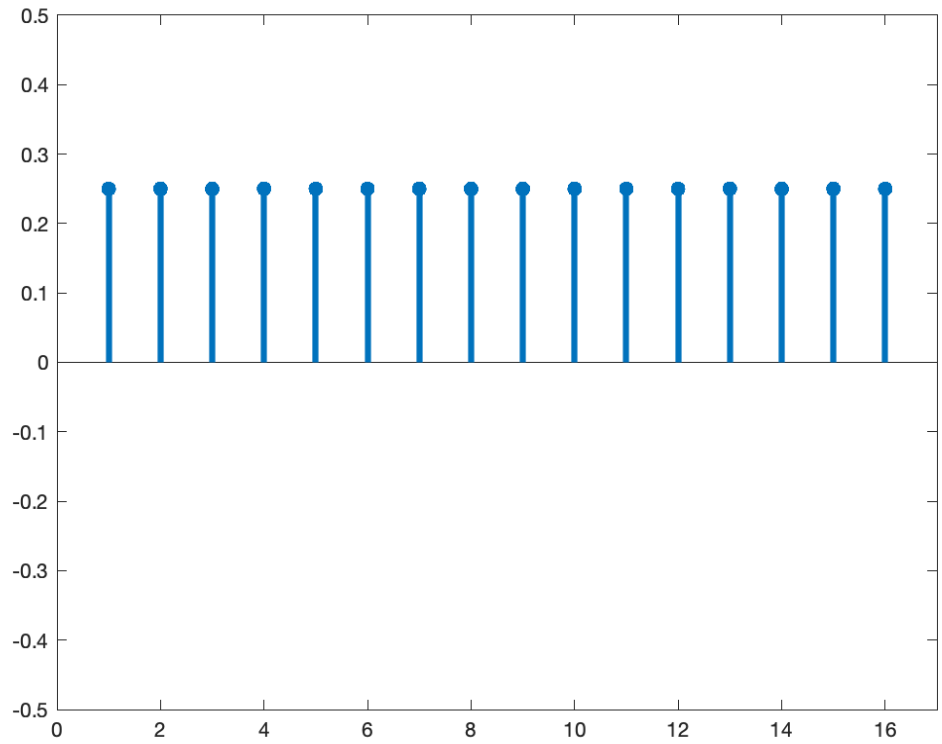


Figure 22: 16-point basis functions of discrete Cosine transform

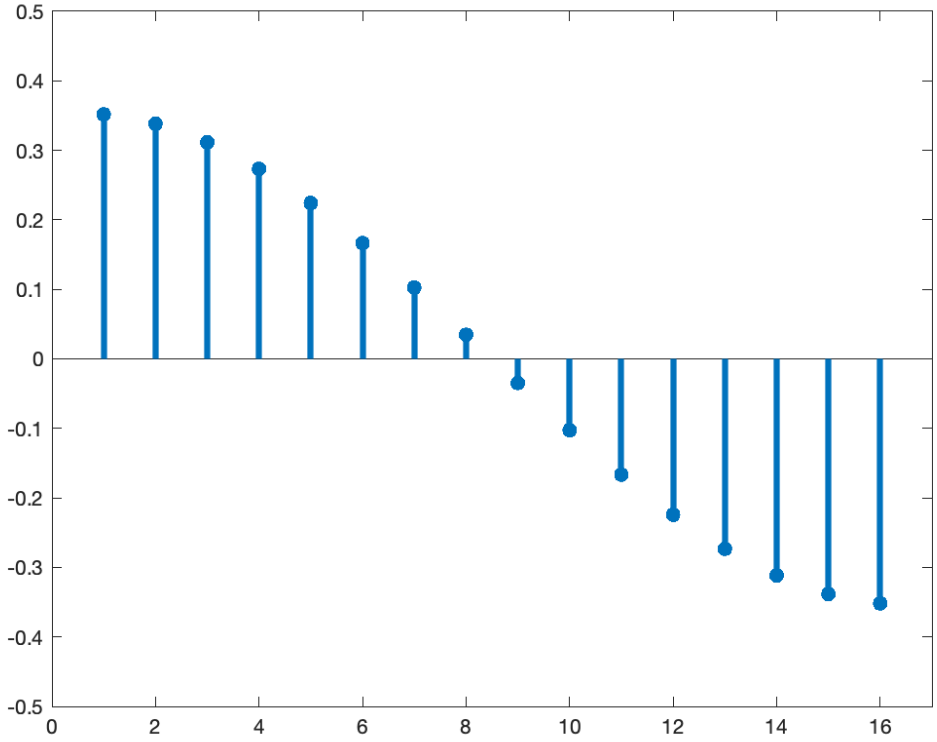


Figure 23: 16-point basis functions of discrete Cosine transform

Let $N = 16$, $k = 1$ and $n = 0$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(0)+1)1}{2(16)}\right) = 0.3519$.

Let $N = 16$, $k = 1$ and $n = 1$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(1)+1)1}{2(16)}\right) = 0.3383$.

Let $N = 16$, $k = 1$ and $n = 2$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(2)+1)1}{2(16)}\right) = 0.3118$.

Let $N = 16$, $k = 1$ and $n = 3$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(3)+1)1}{2(16)}\right) = 0.2733$.

Let $N = 16$, $k = 1$ and $n = 4$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(4)+1)1}{2(16)}\right) = 0.2243$.

Let $N = 16$, $k = 1$ and $n = 5$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(5)+1)1}{2(16)}\right) = 0.1667$.

Let $N = 16$, $k = 1$ and $n = 6$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(6)+1)1}{2(16)}\right) = 0.1026$.

Let $N = 16$, $k = 1$ and $n = 7$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(7)+1)1}{2(16)}\right) = 0.0347$.

Let $N = 16$, $k = 1$ and $n = 8$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(8)+1)1}{2(16)}\right) = -0.0347$.

Let $N = 16$, $k = 1$ and $n = 9$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(9)+1)1}{2(16)}\right) = -0.1026$.

Let $N = 16$, $k = 1$ and $n = 10$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(10)+1)1}{2(16)}\right) = -0.1667$.

Let $N = 16$, $k = 1$ and $n = 11$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(11)+1)1}{2(16)}\right) = -0.2243$.

Let $N = 16$, $k = 1$ and $n = 12$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(12)+1)1}{2(16)}\right) = -0.2733$.

Let $N = 16$, $k = 1$ and $n = 13$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(13)+1)1}{2(16)}\right) = -0.3118$.

Let $N = 16$, $k = 1$ and $n = 14$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(14)+1)1}{2(16)}\right) = -0.3383$.

Let $N = 16$, $k = 1$ and $n = 15$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(15)+1)1}{2(16)}\right) = -0.3519$.

Now considering Figure 21 (c) and the equivalent Figure 24 the equations will be calculated allowing $N = 16, k = 2$, and $0 \leq n \leq N-1$ resulting in Equation 71 being calculated as follows:

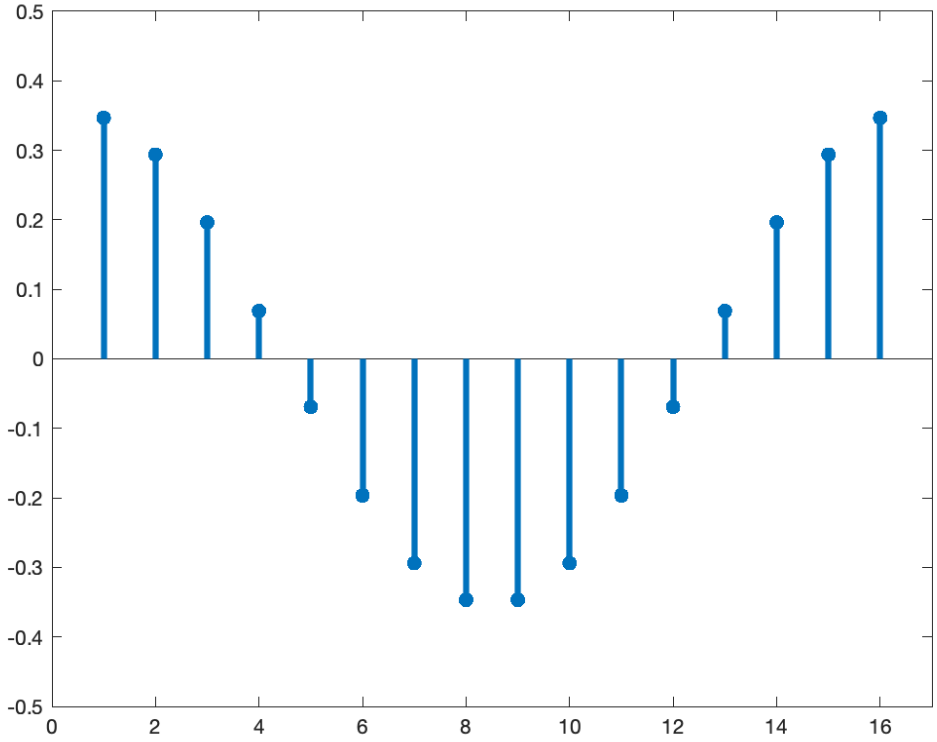


Figure 24: 16-point basis functions of discrete Cosine transform

Let $N = 16$, $k = 2$ and $n = 0$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(0)+1)2}{2(16)}\right) = 0.3468$.

Let $N = 16$, $k = 2$ and $n = 1$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(1)+1)2}{2(16)}\right) = 0.2940$.

Let $N = 16$, $k = 2$ and $n = 2$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(2)+1)2}{2(16)}\right) = 0.1964$.

Let $N = 16$, $k = 2$ and $n = 3$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(3)+1)2}{2(16)}\right) = 0.0690$.

Let $N = 16$, $k = 2$ and $n = 4$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(4)+1)2}{2(16)}\right) = -0.0690$.

Let $N = 16$, $k = 2$ and $n = 5$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(5)+1)2}{2(16)}\right) = -0.1964$.

Let $N = 16$, $k = 2$ and $n = 6$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(6)+1)2}{2(16)}\right) = -0.2940$.

Let $N = 16$, $k = 2$ and $n = 7$will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(7)+1)2}{2(16)}\right) = -0.3468$.

Let $N = 16$, $k = 2$ and $n = 8$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(8)+1)2}{2(16)}\right) = -0.3468$.

Let $N = 16$, $k = 2$ and $n = 9$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(9)+1)2}{2(16)}\right) = -0.2940$.

Let $N = 16$, $k = 2$ and $n = 10$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(10)+1)2}{2(16)}\right) = -0.1964$.

Let $N = 16$, $k = 2$ and $n = 11$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(11)+1)2}{2(16)}\right) = -0.0690$.

Let $N = 16$, $k = 2$ and $n = 12$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(12)+1)2}{2(16)}\right) = 0.0690$.

Let $N = 16$, $k = 2$ and $n = 13$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(13)+1)2}{2(16)}\right) = 0.1964$.

Let $N = 16$, $k = 2$ and $n = 14$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(14)+1)2}{2(16)}\right) = 0.2940$.

Let $N = 16$, $k = 2$ and $n = 15$ will result in Eq. 71 being as follows $\frac{1}{\sqrt{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) = \frac{1}{\sqrt{16}} \cos\left(\frac{\pi(2(15)+1)2}{2(16)}\right) = 0.3468$.

Continuing this process will produce the results for the remaining subfigures in Figure 21. This process will result in Eq. 71 producing the 2-D DCT matrix as follows:

$$C(n,k) = \begin{bmatrix} C(n,k_1) & C(n,k_2) \end{bmatrix} \tag{80}$$

$$C(n,k_1) = \begin{bmatrix} 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 \\ 0.3519 & 0.3383 & 0.3118 & 0.2733 & 0.2243 & 0.1667 & 0.1026 & 0.0347 \\ 0.3468 & 0.2940 & 0.1964 & 0.0690 & -0.0690 & -0.1964 & -0.2940 & -0.3468 \\ 0.3383 & 0.2243 & 0.0347 & -0.1667 & -0.3118 & -0.3519 & -0.2733 & -0.1026 \\ 0.3266 & 0.1353 & -0.1353 & -0.3266 & -0.3266 & -0.1353 & 0.1353 & 0.3266 \\ 0.3118 & 0.0347 & -0.2733 & -0.3383 & -0.1026 & 0.2243 & 0.3519 & 0.1667 \\ 0.2940 & -0.0690 & -0.3468 & -0.1964 & 0.1964 & 0.3468 & 0.0690 & -0.2940 \\ 0.2733 & -0.1667 & -0.3383 & 0.0347 & 0.3519 & 0.1026 & -0.3118 & -0.2243 \\ 0.2500 & -0.2500 & -0.2500 & 0.2500 & 0.2500 & -0.2500 & -0.2500 & 0.2500 \\ 0.2243 & -0.3118 & -0.1026 & 0.3519 & -0.0347 & -0.3383 & 0.1667 & 0.2733 \\ 0.1964 & -0.3468 & 0.0690 & 0.2940 & -0.2940 & -0.0690 & 0.3468 & -0.1964 \\ 0.1667 & -0.3519 & 0.2243 & 0.1026 & -0.3383 & 0.2733 & 0.0347 & -0.3118 \\ 0.1353 & -0.3266 & 0.3266 & -0.1353 & -0.1353 & 0.3266 & -0.3266 & 0.1353 \\ 0.1026 & -0.2733 & 03519 & -0.3118 & 0.1667 & 0.0347 & -0.2243 & 0.3383 \\ 0.0690 & -0.1964 & 0.2940 & -0.3468 & 0.3468 & -0.2940 & 0.1964 & -0.0690 \\ 0.0347 & -0.1026 & 0.1667 & -0.2243 & 0.2733 & -0.3118 & 0.3383 & -0.3519 \end{bmatrix} \tag{81}$$

$$C(n,k_2) = \begin{bmatrix} 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 & 0.2500 \\ -0.0347 & -0.1026 & -0.1667 & -0.2243 & -0.2733 & -0.3118 & -0.3383 & -0.3519 \\ -0.3468 & -0.2940 & -0.1964 & -0.690 & 0.0690 & 0.1964 & 0.2940 & 0.3468 \\ 0.1026 & 0.2733 & 0.3519 & 0.3118 & 0.1667 & -0.0347 & -0.2243 & -0.3383 \\ 0.3266 & 0.1353 & -0.1353 & -0.3266 & -0.3266 & -0.1353 & 0.1353 & 0.3266 \\ -0.1667 & -0.3519 & -0.2243 & 0.1026 & 0.3383 & 0.2733 & -0.0347 & -0.3118 \\ -0.2940 & 0.0690 & 0.3468 & 0.1964 & -0.1964 & -0.3468 & -0.0690 & 0.2940 \\ 0.2243 & 0.3118 & -0.1026 & -0.3519 & -0.0347 & 0.3383 & 0.1667 & -0.2733 \\ 0.2500 & -0.2500 & -0.2500 & 0.2500 & 0.2500 & -0.2500 & -0.2500 & 0.2500 \\ -0.2733 & -0.1667 & 0.3383 & 0.0347 & -0.3519 & 0.1026 & 0.3118 & -0.2243 \\ -0.1964 & 0.3468 & -0.0690 & -0.2940 & 0.2940 & 0.0690 & -0.3468 & 0.1964 \\ 0.3118 & -0.0347 & -0.2733 & 0.3383 & -0.1026 & -0.2243 & 0.3519 & -0.1667 \\ 0.1353 & -0.3266 & 0.3266 & -0.1353 & -0.1353 & 0.3266 & -0.3266 & 0.1353 \\ -0.3383 & 0.2243 & -0.0347 & -0.1667 & 0.3118 & -0.3519 & 0.2733 & -0.1026 \\ -0.0690 & 0.1964 & -0.2940 & 0.3468 & -0.3468 & 0.2940 & -0.1964 & 0.0690 \\ 0.3519 & -0.3383 & 0.3118 & -0.2733 & 0.2243 & -0.1667 & 0.1026 & -0.0347 \end{bmatrix} \tag{82}$$

8.3.1 Matlab Code

In order to create the 16-point DCT matrix, we used the Matlab code shown in Algorithm 10.

8.4 Numerical Example

In this section an example is presented in detail for the number 1 followed by a simple example for the number 7. We will look at the transformation from an image to a DCT and the transformation back to the original image.

8.4.1 Number 1 Example

Figure 25 shows the original image of the number 1 on a 28x28 grid found as index 946 of the numerical example data set.

Figure 27 is Figure 25 resized into a 16x16 grid.

Figure 29 is similar to Figure 27, but Figure 25 is instead resized into a 8x8 grid.

Figures 26, 28, and 30 are the Matlab code used to create the three images.

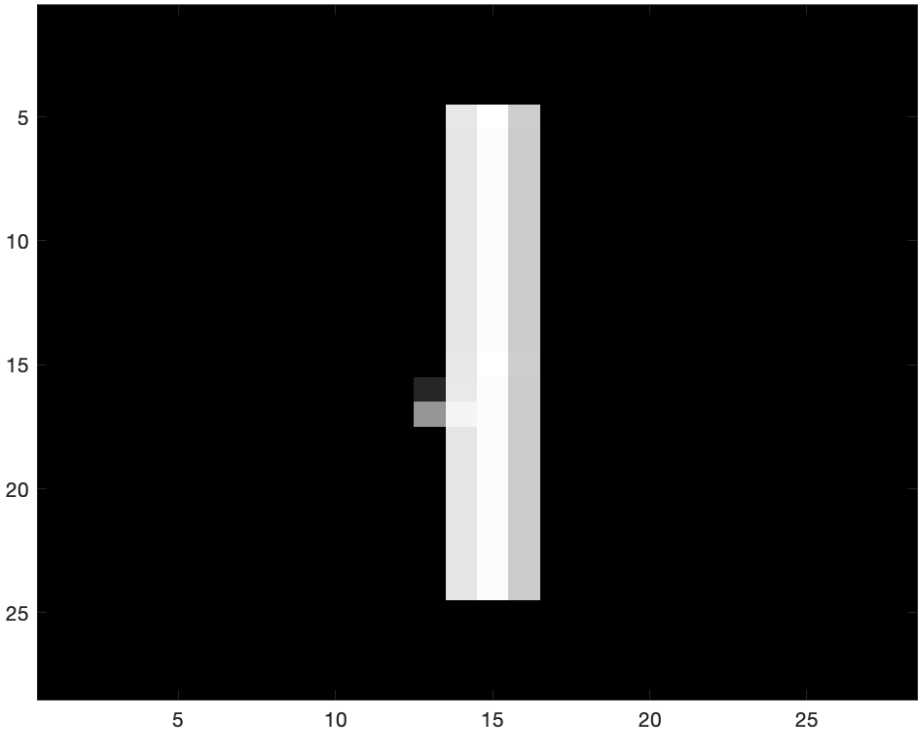


Figure 25: The Number 1 Created in a 28x28 Grid

```
figure,imagesc(reshape(train(946,2:end),[28,28])'),colormap(gray)
```

Figure 26: Matlab code used to create the original image of the number 1

Now let's take advantage of matrix multiplication properties and show an example with the **C** DCT matrix and the image matrix **Im** to show the end result of the transformed image matrix.

It should be noted that the matrix calculated in the following equation creates the image shown in Figure 31, which is an 8x8 DCT of our original resized number 1.

$$\mathbf{D} = (\mathbf{C} \times \mathbf{Im}) \times \mathbf{C}^T = \tag{83}$$

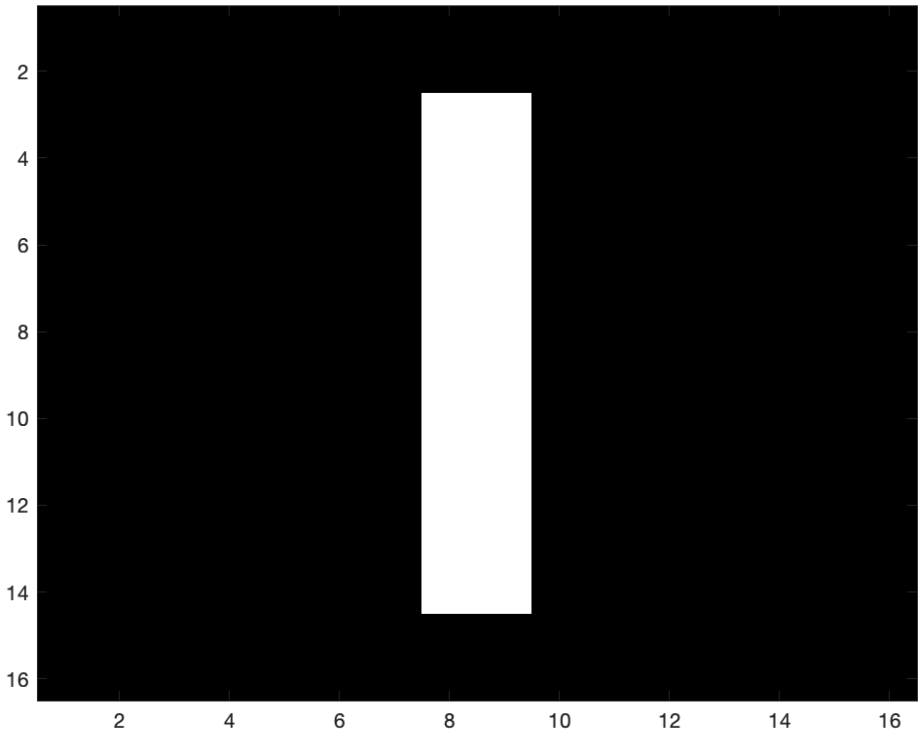


Figure 27: The Number 1 Resized into a 16x16 Grid

```
im16by16_1 = imresize(reshape(train(946,2:end),[28,28])',[16,16]);
for i = 1:16
    for j = 1:16
        if im16by16_1(i,j) < 87
            im16by16_1(i,j) = 0;
        else
            im16by16_1(i,j) = 255;
        end
    end
end
figure,imagesc(im16by16_1),colormap(gray)
```

Figure 28: Matlab code used to create the original image of the number 1

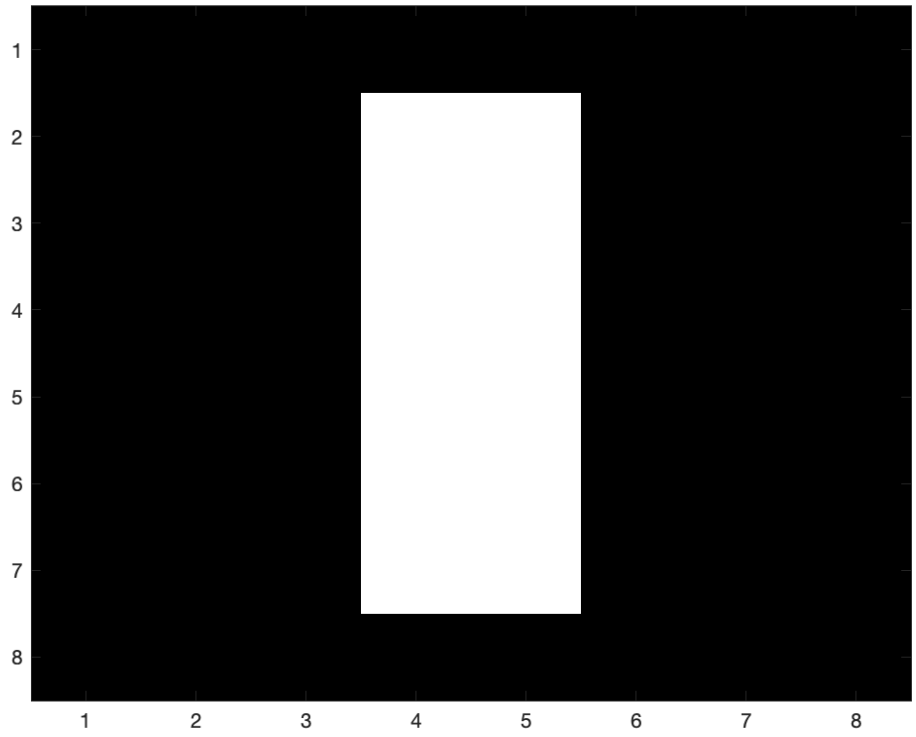


Figure 29: The Number 1 Resized into a 8x8 Grid

```
im8by8_1 = imresize(reshape(train(946,2:end),[28,28])',[8,8]);
for i = 1:8
    for j = 1:8
        if im8by8_1(i,j) < 50
            im8by8_1(i,j) = 0;
        else
            im8by8_1(i,j) = 255;
        end
    end
end
figure,imagesc(im8by8_1),colormap(gray)
```

Figure 30: Matlab code used to create the original image of the number 1

$$\begin{bmatrix} 382.5000 & 0.0000 & -499.7603 & 0.0000 & 382.5000 & 0.0000 & -207.0075 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -166.5868 & 0.0000 & 217.6561 & 0.0000 & -166.5868 & 0.0000 & 90.1561 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -127.5000 & 0.0000 & 166.5868 & 0.0000 & -127.5000 & 0.0000 & 69.0025 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -69.0025 & 0.0000 & 90.1561 & 0.0000 & -69.0025 & 0.0000 & 37.3439 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & -0.0000 \end{bmatrix}$$

where $\mathbf{C}$, $\mathbf{Im}$, $\mathbf{C}^T$, and $(\mathbf{C} \times \mathbf{Im})$ are as follows:

$$\mathbf{C} = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix}$$

$$\mathbf{IM} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{C}^T = \begin{bmatrix} 0.3536 & 0.4904 & 0.4619 & 0.4157 & 0.3536 & 0.2778 & 0.1913 & 0.0975 \\ 0.3536 & 0.4157 & 0.1913 & -0.0975 & -0.3536 & -0.4904 & -0.4619 & -0.2778 \\ 0.3536 & 0.2778 & -0.1913 & -0.4904 & -0.3536 & 0.0975 & 0.4619 & 0.4157 \\ 0.3536 & 0.0975 & -0.4619 & -0.2778 & 0.3536 & 0.4157 & -0.1913 & -0.4904 \\ 0.3536 & -0.0975 & -0.4619 & 0.2778 & 0.3536 & -0.4157 & -0.1913 & 0.4904 \\ 0.3536 & -0.2778 & -0.1913 & 0.4904 & -0.3536 & -0.0975 & 0.4619 & -0.4157 \\ 0.3536 & -0.4157 & 0.1913 & 0.0975 & -0.3536 & 0.4904 & -0.4619 & 0.2778 \\ 0.3536 & -0.4904 & 0.4619 & -0.4157 & 0.3536 & -0.2778 & 0.1913 & -0.0975 \end{bmatrix}$$

$$(\mathbf{C} \times \mathbf{Im}) = \begin{bmatrix} 0 & 0 & 0 & 540.9367 & 540.9367 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & -235.5893 & -235.5893 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & -180.3122 & -180.3122 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & -97.5843 & -97.5843 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0000 & 0.0000 & 0 & 0 & 0 \end{bmatrix}$$

It should be noted that the matrix calculated in the following equation creates the image shown in Figure 33, which is the exact same as the original image in Figure 29.

$$\mathbf{Im} = (\mathbf{C}^T \times \mathbf{D}) \times \mathbf{C} = \tag{84}$$

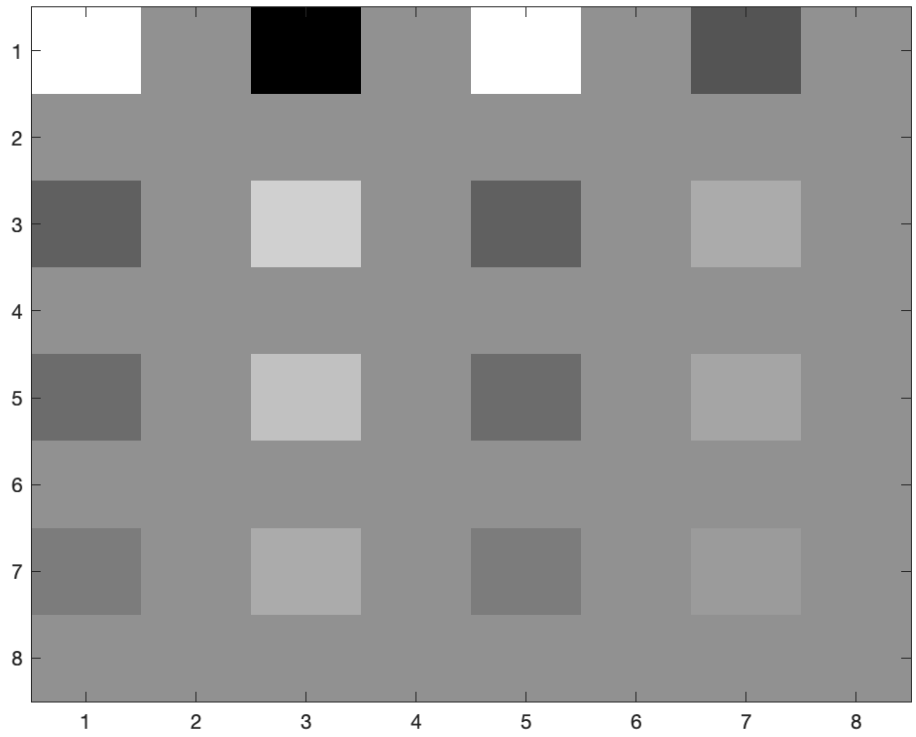


Figure 31: The DCT of the 8x8 number 1

```
C = NPointDCT2(8);
D_1 = C*im8by8_1*C';
figure,imagesc(D_1),colormap(gray)
```

Figure 32: Matlab code used to calculate the 8x8 matrix and create the DCT image

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $\mathbf{C}^T$, $\mathbf{D}$, $\mathbf{C}$, and $(\mathbf{C}^T \times \mathbf{D})$ are as follows:

$$\mathbf{C}^T = \begin{bmatrix} 0.3536 & 0.4904 & 0.4619 & 0.4157 & 0.3536 & 0.2778 & 0.1913 & 0.0975 \\ 0.3536 & 0.4157 & 0.1913 & -0.0975 & -0.3536 & -0.4904 & -0.4619 & -0.2778 \\ 0.3536 & 0.2778 & -0.1913 & -0.4904 & -0.3536 & 0.0975 & 0.4619 & 0.4157 \\ 0.3536 & 0.0975 & -0.4619 & -0.2778 & 0.3536 & 0.4157 & -0.1913 & -0.4904 \\ 0.3536 & -0.0975 & -0.4619 & 0.2778 & 0.3536 & -0.4157 & -0.1913 & 0.4904 \\ 0.3536 & -0.2778 & -0.1913 & 0.4904 & -0.3536 & -0.0975 & 0.4619 & -0.4157 \\ 0.3536 & -0.4157 & 0.1913 & 0.0975 & -0.3536 & 0.4904 & -0.4619 & 0.2778 \\ 0.3536 & -0.4904 & 0.4619 & -0.4157 & 0.3536 & -0.2778 & 0.1913 & -0.0975 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 382.5000 & 0.0000 & -499.7603 & 0.0000 & 382.5000 & 0.0000 & -207.0075 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -166.5868 & 0.0000 & 217.6561 & 0.0000 & -166.5868 & 0.0000 & 90.1561 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -127.5000 & 0.0000 & 166.5868 & 0.0000 & -127.5000 & 0.0000 & 69.0025 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -69.0025 & 0.0000 & 90.1561 & 0.0000 & -69.0025 & 0.0000 & 37.3439 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & -0.0000 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix}$$

$$(\mathbf{C}^T \times \mathbf{D}) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 180.3122 & 0.0000 & -235.5893 & 0.0000 & 180.3122 & 0.0000 & -97.5843 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

8.4.2 Number 7 Example

This transform can also be seen through an 8x8 image of the number 7.

$$\mathbf{D} = (\mathbf{C} \times \mathbf{Im}) \times \mathbf{C}^T = \tag{85}$$

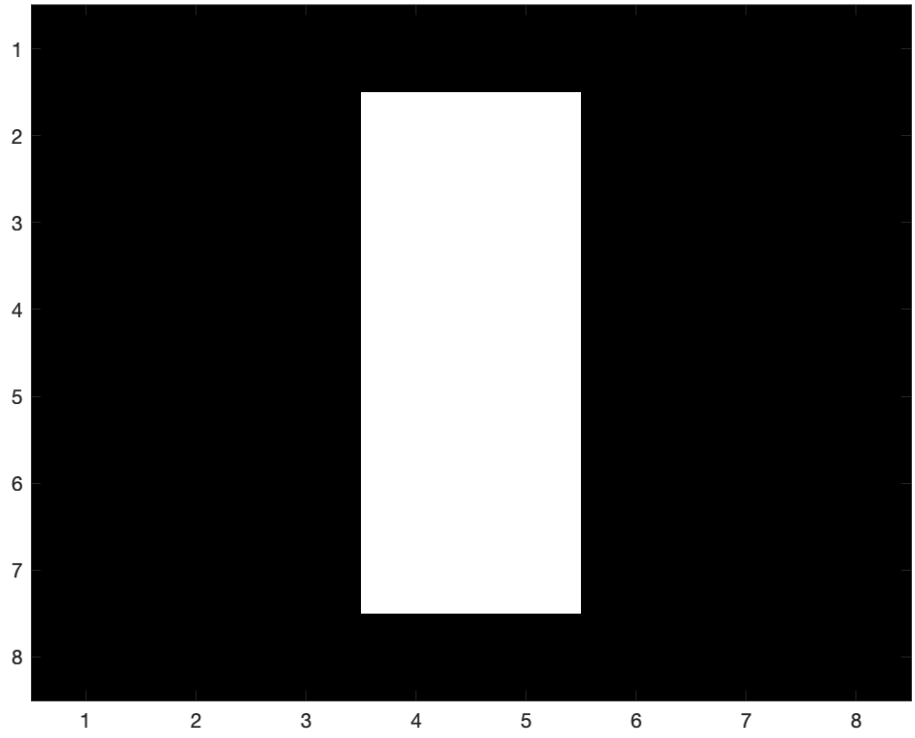


Figure 33: The Number 1 in an 8x8 grid

$$\begin{bmatrix} 382.5000 & -21.4013 & -343.2787 & 104.6735 & 63.7500 & -99.2815 & 4.1857 & -55.7942 \\ -27.7922 & 26.5866 & 120.9233 & 1.2408 & -178.0563 & -46.9351 & 88.4238 & -5.2884 \\ -152.2959 & 72.8986 & 76.9531 & -189.7687 & 117.7946 & 165.1352 & -122.0311 & -27.9622 \\ -171.6960 & -81.4363 & 74.9621 & 96.0607 & 93.5754 & -64.1857 & -36.6263 & 182.7855 \\ -127.5000 & -78.7748 & 107.6895 & 72.1741 & -63.7500 & 64.1043 & 93.3986 & -94.1299 \\ 61.0759 & 16.1987 & -50.0881 & -19.1077 & 8.3101 & 12.7673 & -7.2854 & -36.3583 \\ 132.0855 & 74.1074 & -58.2811 & -53.7308 & -48.7921 & -56.6489 & -13.2031 & 17.7586 \\ 235.4298 & 39.7897 & -180.9744 & 1.8570 & 10.5436 & -70.2433 & -17.5886 & -7.9147 \end{bmatrix}$$

where $\mathbf{C}$, $\mathbf{Im}$, $\mathbf{C}^T$, and $(\mathbf{C} \times \mathbf{Im})$ are as follows:

$$\mathbf{C} = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix}$$

$$\mathbf{IM} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 255 & 255 & 255 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{C}^T = \begin{bmatrix} 0.3536 & 0.4904 & 0.4619 & 0.4157 & 0.3536 & 0.2778 & 0.1913 & 0.0975 \\ 0.3536 & 0.4157 & 0.1913 & -0.0975 & -0.3536 & -0.4904 & -0.4619 & -0.2778 \\ 0.3536 & 0.2778 & -0.1913 & -0.4904 & -0.3536 & 0.0975 & 0.4619 & 0.4157 \\ 0.3536 & 0.0975 & -0.4619 & -0.2778 & 0.3536 & 0.4157 & -0.1913 & -0.4904 \\ 0.3536 & -0.0975 & -0.4619 & 0.2778 & 0.3536 & -0.4157 & -0.1913 & 0.4904 \\ 0.3536 & -0.2778 & -0.1913 & 0.4904 & -0.3536 & -0.0975 & 0.4619 & -0.4157 \\ 0.3536 & -0.4157 & 0.1913 & 0.0975 & -0.3536 & 0.4904 & -0.4619 & 0.2778 \\ 0.3536 & -0.4904 & 0.4619 & -0.4157 & 0.3536 & -0.2778 & 0.1913 & -0.0975 \end{bmatrix}$$

$$(\mathbf{C} \times \mathbf{Im}) = \begin{bmatrix} 0 & 90.1561 & 90.1561 & 270.4683 & 360.6245 & 270.4683 & 0 & 0 \\ 0 & 70.8352 & 70.8352 & -160.2273 & -130.8864 & 70.8352 & 0 & 0 \\ 0 & -48.7921 & -48.7921 & 117.7946 & -166.5868 & -284.3814 & 0 & 0 \\ 0 & -125.0501 & -125.0501 & -206.1885 & 95.7092 & -125.0501 & 0 & 0 \\ 0 & -90.1561 & -90.1561 & -90.1561 & -180.3122 & 90.1561 & 0 & 0 \\ 0 & 24.8740 & 24.8740 & 79.0889 & 19.0377 & 24.8740 & 0 & 0 \\ 0 & 117.7946 & 117.7946 & 48.7921 & 69.0025 & 20.2104 & 0 & 0 \\ 0 & 106.0124 & 106.0124 & 151.9736 & 195.8853 & 106.0124 & 0 & 0 \end{bmatrix}$$

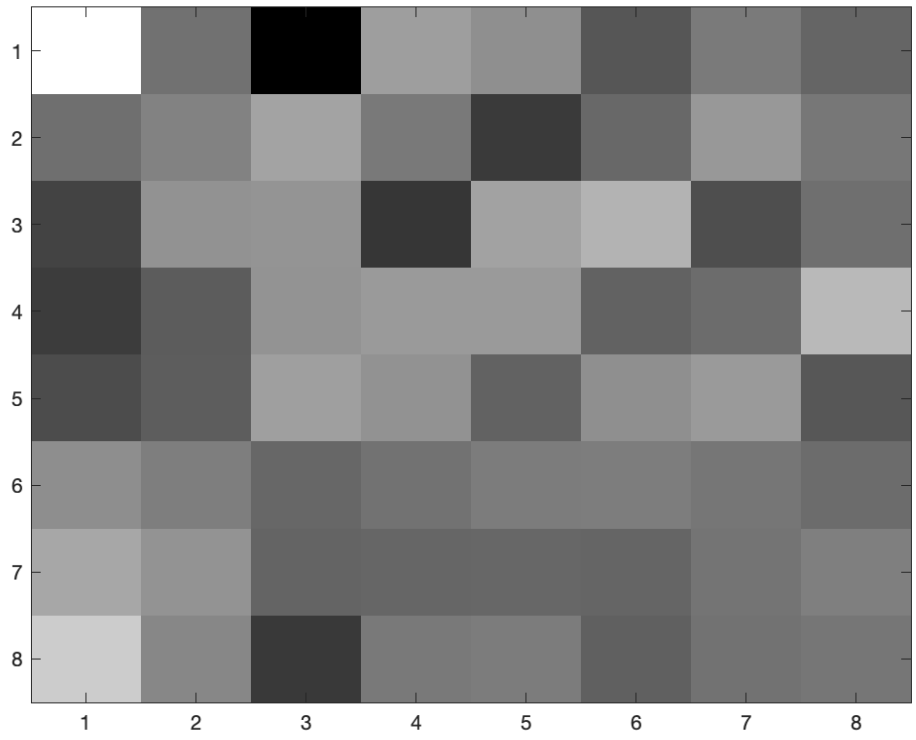


Figure 34: The DCT of the 8x8 number 7

9 Generating Features

In generating features it is important to understand the data being processed, images, video, audio, acoustic, radiometric, sonar, medical, banking, etc. In each of these data types the data can also be represented in various forms. Let consider an audio file, it can be represented as analog, various bits, frequencies, collection microphones, stereo, Dolby Digital, communication channel, etc. Based on the data type, the transformation needed will determine how well the features will represent the data in a compressed fashion. Let's consider taking the FFT of a standard 44.1kHz musical audio signal the vocals can be represented with a few real and imaginary coefficients. For images as shown with the DCT transform of the numerical examples you can subdivide the image into the directional coefficients, e.g. vertical, horizontal and diagonal. From there you can perform first order statistics or eigendecomposition.

9.1 Statistical methods (mean, moment, standard deviation, skewness, kurtosis)

When studying first order statistics, the use is clear in understanding the data mean, standard deviation and other statistics. By using the first order statistics an initial assessment give the data analyst a great deal of information. With the use of transforms (DFT, DCT, Wavelets, etc.) the statistical methods can be used to extract characteristics the transforms extract. The statistics calculated over the data vectors $\mathbf{x} = x_i = [x_1, x_2, \ldots, x_n] \in X_n$ are used to generate the features that represent the transformed raw data in a much small form. The table below lists five statistics: mean, standard deviation, skewness, and kurtosis along with their calculation.

Table 2: Statistics for Generating Features

Test Statistics	**Statistical Function $\boldsymbol{F}(\cdot)$**
Mean	$F_\mu(\mathbf{x}) = \mu(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$
Standard Deviation	$F_\sigma(\mathbf{x}) = \left(\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu(\mathbf{x})\right)^2\right)^{1/2}$
Skewness	$F_\gamma(\mathbf{x}) = \gamma(\mathbf{x}) = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu(\mathbf{x})\right)^3}{\sigma(\mathbf{x})^3}$
Kurtosis	$F_\kappa(\mathbf{x}) = \kappa(\mathbf{x}) = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu(\mathbf{x})\right)^4}{\sigma(\mathbf{x})^4}$

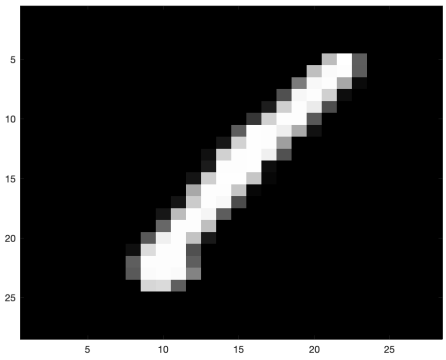
The use of both transforms and statistics allow a large data set to be reduce to a smaller set of data that is an order or two orders of magnitude smaller.

Figure 35 shows the numbers and the corresponding DCT. It should be noted how the diagonal and horizontal characteristics of the numerical data shows up on the 2-dimensional DCT images. Making a comparison with Figure 15, specifically, subfigures d, e, f, g, h and i, it can be seen that the vertical, horizontal and diagonal characteristics of the images come through clearly.

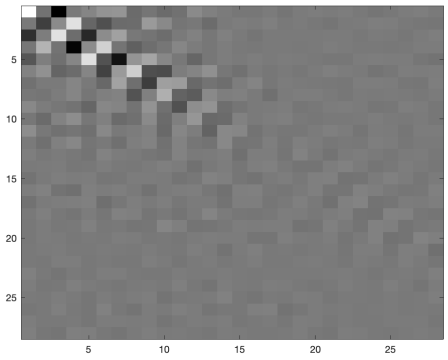
9.2 Eigendecomposition

Another approach to generate a reduced set of feature from the input features is to use a transformed space instead of the original feature space, e.g., the numerical data set. For example using a transformation $f(x)$ that maps the data points $\mathbf{x}$ of the input space, $x[n]$, into a reduced dimensional space $x'[p]$, where $n > p$, creates features in a new space that may have better discriminatory properties. Classification is based on the new feature space rather than the input feature space.

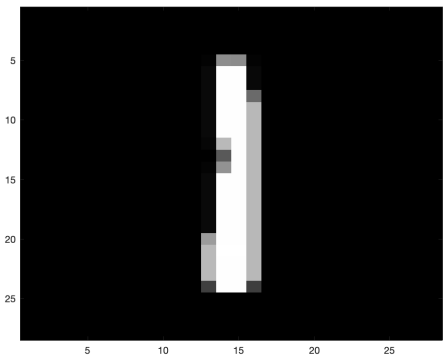
Previously in Section 5 eigendecomposition was used to generate features, in this section the reduction in dimensionality can also be accomplished using eigenvectors. How ever the eignevectors alone may not be sufficient,



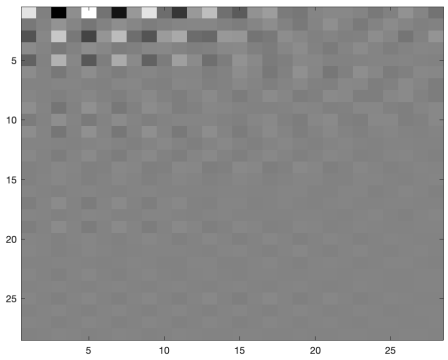
(a) Number 1 Index 1



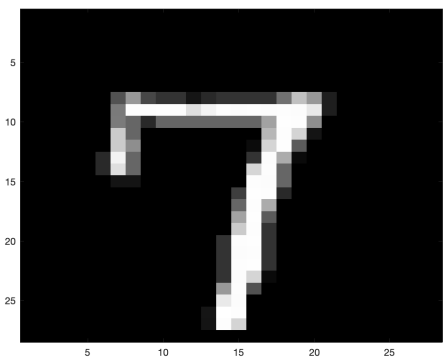
(b) Corresponding DCT



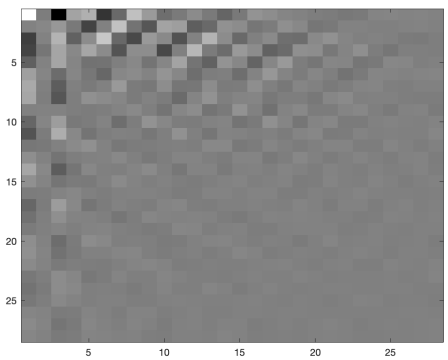
(c) Number 1 Index 3



(d) Corresponding DCT



(e) Number 7 Index 7



(f) Corresponding DCT

Figure 35: Numerical Images and corresponding DCT (a) image number 1 index 1 (b) corresponding DCT (c) image number 1 index 3 (d) corresponding DCT (e) image number 7 index 7 (f) corresponding DCT.

it may require the used of factor analysis in which the factors are rotated in order to group the appropriate variables based on their correlations. Examples will be given, however the exact details are outside of the scope of this document. In the Machine Learning I module linear discriminant analysis will be described which will describe dimensionality reduction in more detail using eigendeconposition.

9.3 Example

1. Take the 2 dimensional Discrete Cosine Transform (DCT) of each matrix from the MNIST numerical data, the matrix represents each number.
2. Extract the vertical, horizontal and diagonal coefficients from the transform.
3. For each of the three sets of DCT coefficients perform eigendecomposition.
4. Retain either the top n number of eigenvectors or the top eigenvectors with maximum variance.
5. Using your top eigenvectors reduce the DCT transformed data. This will create a new data set that represents
6. Now you can save the new features that represent the numerical values.

This example is shown using Matlab code as shown in Algorithm 11 to show the steps of how raw image data can be take and processed using two well know transformation methods to generate features that represent the image with a significantly small set of features when compared to the number of pixels. This code also produces two figures where one of the figures is shown in Figure 36 as well as writing out the features into an Excel file.

Algorithm 11 Matlab Code: Numerical Feature Generating Example using DCT and Eigendecomposition

```
% This code is for educational and research purposes of comparisons. This
% is a numerical reader and processing for generating features. The end
% result will be a 10 class number data set where each observation will
% have fewer features from the 28x28 pixel size images. The example
% shows that 60 features can represent the 28x28 images (784 pixels).
%
% References:
%    https://www.kaggle.com/c/digit-recognizer/data

clear;
clc;
close all;

train = readmatrix('train.csv');
numbObs = 1000;

   % Get the class labels for all observations
classLabels = train(:,1);
   % Get the class labels for the first number of observations of interest
indx0 = find(classLabels(1:numbObs) == 0);
indx1 = find(classLabels(1:numbObs) == 1);
indx2 = find(classLabels(1:numbObs) == 2);
indx3 = find(classLabels(1:numbObs) == 3);
indx4 = find(classLabels(1:numbObs) == 4);
indx5 = find(classLabels(1:numbObs) == 5);
indx6 = find(classLabels(1:numbObs) == 6);
indx7 = find(classLabels(1:numbObs) == 7);
indx8 = find(classLabels(1:numbObs) == 8);
```

```
indx9 = find(classLabels(1:numbObs) == 9);

    % Identify the mask that will retain the diagonal, verticla and
    % horizontal coefficients
diagMask = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0;
            0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];
indxD = find(diagMask == 1);
vertMask = [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

```matlab
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
indxV = find(vertMask == 1);
horizMask = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
             1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
indxH = find(horizMask == 1);

   % Display the masks for visual inspection
figure,imagesc(diagMask), colormap('gray')
figure,imagesc(vertMask), colormap('gray')
figure,imagesc(horizMask), colormap('gray')

   % Create empty data structures to store the image coefficients
dataD = [];
dataV = [];
dataH = [];

   % Loop though the number of selected observations, perform the DCT,
   % followed by storing the coefficients into a matrix for processing
   % later.
for i = 1:numbObs
```

```
135     img = reshape(train(i,2:end),[28,28])';
136     imgDCT = dct2(img);
137
138     dataD = [dataD imgDCT(indxD)];
139     dataV = [dataV imgDCT(indxV)];
140     dataH = [dataH imgDCT(indxH)];
141 end
142
143    % Begine the eigendecomposition process. Zero mean the data, calcualte
144    % the covariance matrix and finally extract the eigenvectors and
145    % eigenvalues.
146 [VD, E] = eig(cov(dataD'- mean(dataD')));
147 [VV, E] = eig(cov(dataV'- mean(dataV')));
148 [VH, E] = eig(cov(dataH'- mean(dataH')));
149    % After evaluating the eigenvalues in E it is seen that diagnal values
150    % are in assending order so flip the corresponsing eigne vectors to be
151    % the decending order of importance (variance high to low).
152 uD = fliplr(VD); % this simply flips the matrix from left to right
153 uV = fliplr(VV);
154 uH = fliplr(VH);
155    % Now we use the top 20 eigenvectors from above to generate the features
156 features = [];
157 features = [features dataD'*uD(:,1:20)];
158 features = [features dataV'*uV(:,1:20)];
159 features = [features dataH'*uH(:,1:20)];
160    % Plot the data and color code each of the numbers to view the
161    % differences.
162 figure,plot(features(indx0,3), features(indx0,21),'or',...
163         features(indx1,3), features(indx1,21),'+g',...
164         features(indx2,3), features(indx2,21),'*b',...
165         features(indx3,3), features(indx3,21),'.k',...
166         features(indx4,3), features(indx4,21),'xm',...
167         features(indx5,3), features(indx5,21),'sr',...
168         features(indx6,3), features(indx6,21),'dg',...
169         features(indx7,3), features(indx7,21),'^b',...
170         features(indx8,3), features(indx8,21),'pk',...
171         features(indx9,3), features(indx9,21),'hm')
172 legend('0','1','2','3','4','5','6','7','8','9')
173 axis([-1000 1000 -1000 1000])
174 title('Numerical data transformed using DCT and Eigendecomposition')
175 xlabel('Feature 3 - 3rd feature from the diagonal coefficients')
176 ylabel('Feature 21 - 1st feature from the vertical coefficients')
177     % Using the top two features 3 and 21 plot the numbers 0, 1, 3 and 4.
178     % The features were identified as being the top features using Fishers
179     % Discriminant Ratio.
180 figure,plot(features(indx0,3), features(indx0,21),'or',...
181         features(indx1,3), features(indx1,21),'+g',...
182         features(indx3,3), features(indx3,21),'.k',...
183         features(indx4,3), features(indx4,21),'xm',...
184                            'LineWidth',2,'MarkerSize',10)
185 legend('0','1','3','4')
186 axis([-1000 1000 -1000 1000])
187 title('Numerical values 0, 1, 3, and 4 transformed using DCT and Eigendecomposition
```

```
    ')
xlabel('Feature 3 - 3rd feature from the diagonal coefficients')
ylabel('Feature 21 - 1st feature from the vertical coefficients')

writematrix([classLabels(1:numbObs) features],'trainFeatures.xls');
```

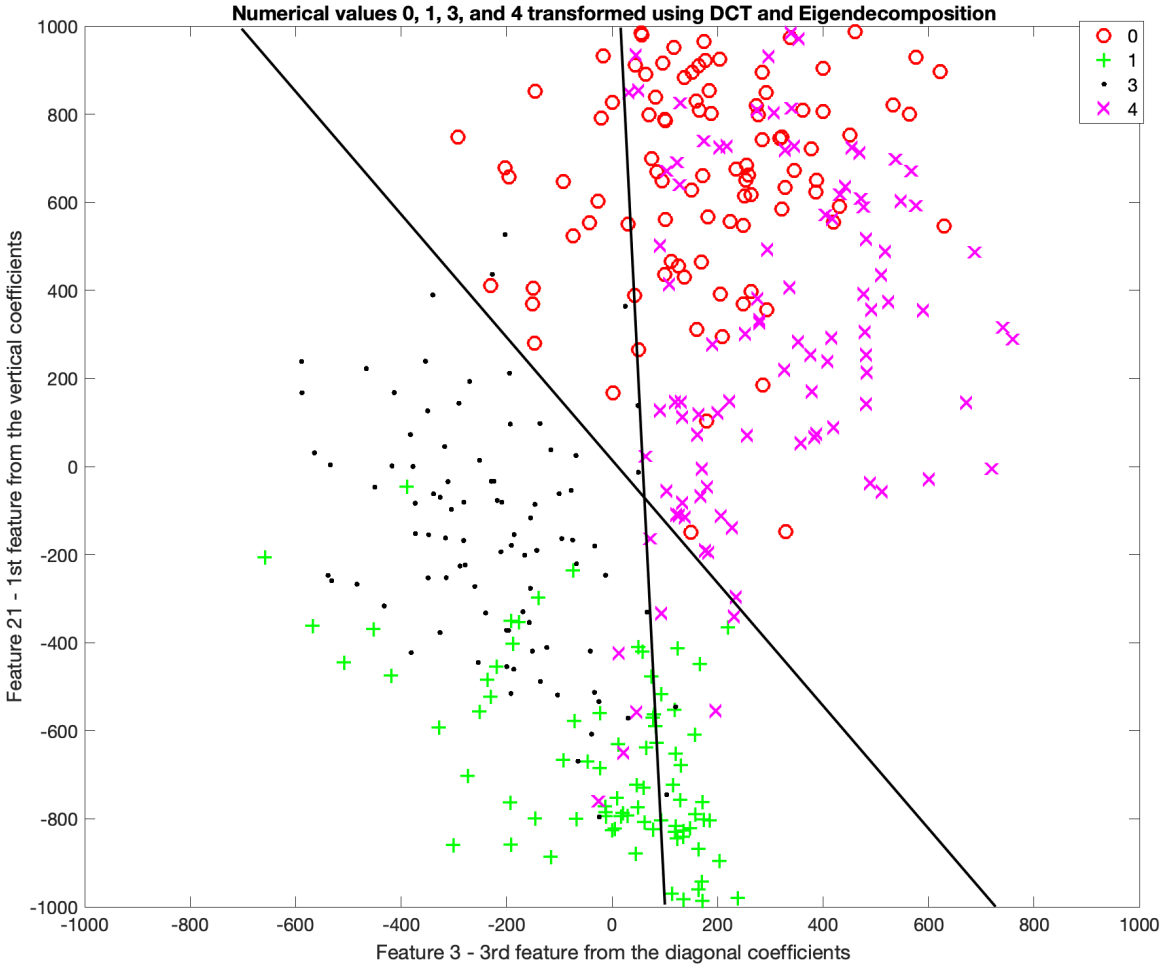


Figure 36: Numerical values 0, 1, 3, and 4 transformed using DCT and Eigendecomposition with the 3rd feature from the diagonal coefficients on the x-axis and the 1st feature from the vertical coefficients.

It should be noted that in Figure 36 the first 1000 observations from the MNIST dte set are used to generate the features from the raw image files. In this figure only the numbers 0, 1, 3 and 4 are represented. Analyzing the figure it can be seen that in 2 dimensions the numbers 0 and 1 have a clear linear boundary line crossing the figure diagonally from the upper left tot the lower right. It should also be noted that the vertical linear line separates the numbers 3 and 4. There are two observations to the right of the line that represent the number three as black points on the figure and 4 magenta x's to the left of the boundary line that represent the number 4. What should be noted that the original images are represented by 784 pixels and in Figure 36 the numerical values are represented

with only two features. Using a feature ranking method along with a nonlinear machine learning algorithm the generated features as small as the top 20 can be used to obtain a 90% classification accuracy.

10 References

[1] Background story: Profile: David A. Huffman, Scientific American, September 1991, pp. 54-58.

[2] Bishop, Christopher M., *Pattern Recognition and Machine Learning*, Springer, 2006

[3] Brown, W. and Shepherd, B. J., *Graphics File Formats: Reference and Guide*, Greenwich, CT: Manning Publications, 1995

[4] Caglayan, O., *Digital Multimedia System Security Based on Transform Domain Techniques*, Dissertation, 2008

[5] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronal L., and Stein, Clifford, *Introduction to Algorithms*, 3rd Edition, MIT Press, 2009

[6] d'Alambert, J.R., *Researchers sur diferentes points importants du systeme du monde*, vol. 2, pp. 66, 1754

[7] *Digit Recognizer*, Kaggle, https://www.kaggle.com/competitions/digit-recognizer/data?select=train.csv

[8] Dillon, and Goldstein, M.. *Multivariate Analysis Methods and Applications*, John Wiley, 1984

[9] Duda, Richard O., Hart, Peter E., and Stork, David G., *Pattern Classification*, 2nd Edition, John Wiley & Sons, Inc., 2001

[10] Duhamel, P., Hollman, H., *Existence of a 2n FFT algorithm with a number of multiplications lower than* 2^{n+1} Electron. Lett., vol 20, pp. 690-692, 1984

[11] Duin, Robert P.W., Tax, David and Pekalska, Elzbieta, *PRTools*, http://prtools.tudelft.nl/

[12] Elysium Ltd., *Home of the JPEG committee—*, Retrieved February 20, 2005, http://www.JPEG.org/, 2004

[13] Euler, L., *Nova Acta Acad. Sci. Petrop.*, v(1), 14, 435-542-84, publ. 1760

[14] Farid, H., *Detecting Hidden Messages Using Higher-Order Statistical Models*, IEEE International Conference on Images Processing, Volume 2, Rochester, NY, pp. 905-908, 2002

[15] Franc, Vojtech and Hlavac, Vaclav, *Statistical Pattern Recognition Toolbox*, https://cmp.felk.cvut.cz/cmp/software/stprtool/index.html

[16] Goldstine, H.H., *A History of Numerical Analysis from the 16th Through the 19th Century*, pp. 249-253, New York, Springer-Verlag, 1977 (See also M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast Fourier transform," IEEE ASSP Magazine 1 (4), 14–21 (1984))

[17] Gonzalez, Rafael C., Woods, Richard E., and Eddins, Steven L., *Digital Image Processing: Using Matlab*, Upper Saddle River, NJ, Pearson Prentice Hall, 2004

[18] Gonzalez, R. C. and Woods R., *Digital Image Processing*, 3rd Edition. Upper Saddle River, NJ: Prentice Hall, 2007

[19] Hotelling, H., *Analysis of a complex of statistical variables into principal components*, Journal of Educational Psychology, Number 24, pp. 417–441, 1933

[20] Huffman, D.A., "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098-1102.

[21] Independent JPEG Group, *Independent JPEG Group*, Retrieved February 20, 2005, http://www.ijg.org/, 1988

[22] JPEG, *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*, ISE/IEC IS 10918-1, American National Standards Institute, Retrieved June 15, 2008 http://www.w3.org/Graphics/JPEG/itu-t81.pdf., 1994

[23] Kaiser, H. F., *The varimax criterion for analytic rotation in factor analysis*, Psychometrika, 23(3), pp. 187-200, 1958

[24] Kaiser, H. F., *The application of electronic computers to factor analysis*, Educational and Psychological Measurement, Volume 20, Number 1, pp. 141-151, 1960

[25] Machine Learning at Waikato University, *WEKA*, https://www.cs.waikato.ac.nz/ ml/index.html

[26] Murry, J. D. and vanRyper, W., *Encyclopedia of Graphics File Formats*, Sebastopol, CA: O'Reilly & Associates, Inc., 1994

[27] Rao, K. P. and Yip, P., *Discrete Cosine Transform Algorithms, Advantages, Applications*, San Diego, CA: Academic Press, Inc., 1990

[28] Theodoridis, S. and Koutroumbas, K., *Pattern Recognition Third Edition*, San Diego, CA: Academic Press, 2006

[29] AstroDave, Will Cukierski, *Digit Recognizer*, 2012, https://kaggle.com/competitions/digit-recognizer

[30] Jackson, Zohar and Souza, César and Flaks, Jason and Pan, Yuxin and Nicolas, Hereman and Thite, Adhish, *Free Spoken Digit Dataset (FSDD)*, 2018, https://www.kaggle.com/datasets/joserzapata/free-spoken-digit-dataset-fsdd

[31] Arne Jense and Anders la Cour-Harbo, "Ripples in Mathematics: the Discrete Wavelet Transform", Springer, 2001

[32] Ian Kaplan, "The Daubechies D4 Wavelet Transform", 2002
http://bearcave.com/software/java/wavelets/daubechies/index.html

[32] Ingrid Daubechies, Ten Lectures on Wavelets, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 61, SIAM, Philadelphia, 1992

[34] Yves Nievergelt, Wavelets Made Easy, BirkHauser, Boston, 1999