UMBC CSEE

40/40

Foundations of Data Mining

CS-691

Homework Assignment - 2

Sandipan Dey

2009

1. (a)

<u>Given</u>,

$$f:\{0,1\}^n \to R$$

$$\psi_{\bar{j}}(\bar{x}) = (-1)^{\bar{j}.\bar{x}}$$

$$\bar{j}, \bar{x} \in \{0,1\}^n$$

$$f(\bar{x}) = \sum_{\bar{j}\in\{0,1\}^n} w_{\bar{j}} \psi_{\bar{j}}(\bar{x}).$$

We first prove that ψ is <u>orthogonal</u>, i.e.,

$$\sum_{\bar{x}\in\{0,1\}^n} \psi_{\bar{i}}(\bar{x}).\psi_{\bar{j}}(\bar{x}) = \begin{Bmatrix} 0, & \bar{i} \neq \bar{j} \\ 2^n, & \bar{i} = \bar{j} \end{Bmatrix} \qquad (1)$$

<u>Proof:</u>

$$\sum_{\bar{x}\in\{0,1\}^n} \psi_{\bar{i}}(\bar{x}).\psi_{\bar{j}}(\bar{x}) = \sum_{\bar{x}\in\{0,1\}^n} (-1)^{\bar{i}.\bar{x}}.(-1)^{\bar{j}.\bar{x}}$$

$$\bar{i} = \bar{j} \Rightarrow \sum_{\bar{x}\in\{0,1\}^n} (-1)^{\bar{i}.\bar{x}+\bar{j}.\bar{x}} = \sum_{\bar{x}\in\{0,1\}^n} (-1)^{2(\bar{i}.\bar{x})} = \sum_{\bar{x}\in\{0,1\}^n} 1 = 2^n$$

$$\bar{i} \neq \bar{j} \Rightarrow \sum_{\bar{x}\in\{0,1\}^n} (-1)^{\bar{i}.\bar{x}+\bar{j}.\bar{x}} = 2^{n-1}(-1)^{odd\ number} + 2^{n-1}(-1)^{even\ number} = 2^{n-1}(1-1) = 0$$

Since $\bar{i}$ and $\bar{j}$ are fixed unequal vectors, $\exists$ at least one bit where they differ. Let's 1st consider that these vectors differ in exactly 1 bit (say k^{th} LSB). Then that particular bit generates a **partition** over the set of vectors $\bar{x} \in \{0,1\}^n$ and divides it into 2 equal disjoint sets:

$\{0,1\}^n = \{\{0,1\}^{n-k-1}0\{0,1\}^{k-1}\} \cup \{\{0,1\}^{n-k-1}1\{0,1\}^{k-1}\}$, with

$$\sum_{\bar{x}\in\{0,1\}^n} (-1)^{\bar{i}.\bar{x}+\bar{j}.\bar{x}} = \sum_{\bar{x}\in\{0,1\}^{n-k-1}0\{0,1\}^{k-1}} (-1)^{\bar{i}.\bar{x}+\bar{j}.\bar{x}} + \sum_{\{0,1\}^{n-k-1}1\{0,1\}^{k-1}} (-1)^{\bar{i}.\bar{x}+\bar{j}.\bar{x}}$$

$$= 2^{n-1}(-1)^{even\ number} + 2^{n-1}(-1)^{odd\ number} = 2^{n-1}(1-1) = 0$$

The 1st exponent is an even number because in that partition $\bar{i}..\bar{x}$ and $\bar{j}..\bar{x}$ are exactly equal, the 2nd exponent is odd since in that partition exactly one of $\bar{i}..\bar{x}$ and $\bar{j}..\bar{x}$ is even, the other one is odd (because exactly one of $\bar{i}$ and $\bar{j}$ has a 1 bit in that position, the other has a 0 bit).

This result can be extended to the general case when the fixed vectors $\bar{i}$ and $\bar{j}$ differ in any arbitrary k bit positions (k = *1..n*), where it generates 2^k equal sized partitions, half of them will have $\bar{i}..\bar{x}+\bar{j}..$

value odd, with other half having the same value even. If $\bar{i}_k$ and $\bar{j}_k$, $k=1..l$ denotes the bits where these vectors differ, we have, $\sum_{k=1}^{l}\bar{i}_k\bar{x}_k + \sum_{k=1}^{l}\bar{j}_k\bar{x}_k = \sum_{k=1}^{l}(\bar{i}_k + \bar{j}_k)\bar{x}_k = \sum_{k=1}^{l}\bar{x}_k$, since $\bar{i}_k + \bar{j}_k = 1$ (the vectors differ in these bits) and their sum is even in all other bits (in which they are exactly equal), which implies $\bar{i}.\bar{x} + \bar{j}.\bar{x}$ is odd or even depending upon whether $\sum_{k=1}^{l}\bar{x}_k$ is odd or even, which happens exactly half of the times, over exactly half of those partitions.

Also, for a fixed $\bar{j}$, $\sum_{\bar{x}\in\{0,1\}^n}\psi_{\bar{j}}(\bar{x}) = \sum_{\bar{x}\in\{0,1\}^n}(-1)^{\bar{j}.\bar{x}} = 2^{n-1}(-1)^{\text{even number}} + 2^{n-1}(-1)^{\text{odd number}} = 0$ (2)

Proof

$\bar{j}$ is fixed. Hence, the total number of cases where $\bar{x}$ matches with $\bar{j}$ in odd number of positions (as a result inner product of them is odd) $= 2^{n-1}$. (They can match exactly in 1 bit position, 3 bit positions, 5 bit positions, ... in $\binom{n}{1}, \binom{n}{3}, \binom{n}{5}, \ldots$ ways respectively, hence total number of ways by which they can differ in odd number of positions $= \binom{n}{1}+\binom{n}{3}+\binom{n}{5}+\ldots$

Arguing in the same manner, hence total number of ways by which they can match in even number of positions (hence inner product is even) $= \binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\binom{n}{6}+\ldots$

Again, putting x = 1 and then $x = -1$ in the Binomial expansion identity

$(1+x)^n = \binom{n}{0}+\binom{n}{1}x+\binom{n}{2}x^2+\binom{n}{3}x^3+\cdots+\binom{n}{n}x^n$, we have,

$$\binom{n}{1}+\binom{n}{3}+\binom{n}{5}+\cdots = \binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\cdots = \frac{\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{n}}{2} = \frac{2^n}{2} = 2^{n-1}$$

Hence, we have,

$$f(\bar{x}) = \sum_{\bar{j} \in \{0,1\}^n} w_{\bar{j}} \psi_{\bar{j}}(\bar{x})$$

$$\Rightarrow f(\bar{x})\psi_{\bar{i}}(\bar{x}) = \sum_{\bar{j} \in \{0,1\}^n} w_{\bar{j}} \psi_{\bar{j}}(\bar{x})\psi_{\bar{i}}(\bar{x})$$

$$\Rightarrow \sum_{\bar{x} \in \{0,1\}^n} f(\bar{x})\psi_{\bar{i}}(\bar{x}) = \sum_{\bar{x} \in \{0,1\}^n} \sum_{\bar{j} \in \{0,1\}^n} w_{\bar{j}} \psi_{\bar{j}}(\bar{x})\psi_{\bar{i}}(\bar{x})$$

$$\Rightarrow \sum_{\bar{x} \in \{0,1\}^n} f(\bar{x})\psi_{\bar{i}}(\bar{x}) = \sum_{\bar{x} \in \{0,1\}^n} w_{\bar{i}} \psi_{\bar{i}}^{\,2}(\bar{x})$$

($\because \psi$ is orthogonal from (1), $\psi_{\bar{j}}(\bar{x})\psi_{\bar{i}}(\bar{x}) \neq 0$ iff $\bar{j} = \bar{i}$ and 0 in all other cases)

$$\Rightarrow \sum_{\bar{x} \in \{0,1\}^n} f(\bar{x})\psi_{\bar{i}}(\bar{x}) = w_{\bar{i}} \sum_{\bar{x} \in \{0,1\}^n} \psi_{\bar{i}}^{\,2}(\bar{x}) = w_{\bar{i}}.2^n$$

$$\Rightarrow w_{\bar{i}} = \frac{1}{2^n} \sum_{\bar{x} \in \{0,1\}^n} f(\bar{x})\psi_{\bar{i}}(\bar{x})$$

Now, with all these ground works, let's proceed towards the actual proof,

we use the following identity:

$$\left(\sum_{i=1}^{n} a_i\right)^2 = \sum_{i=1}^{n} a_i^{\,2} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} a_i a_j = \sum_{i=1}^{n} a_i^{\,2} + 2\sum_{i=1}^{n} \sum_{\substack{j=1 \\ i < j}}^{n} a_i a_j$$

Proof

$$\left(\sum_{\bar{x}} f(\bar{x})\psi_{\bar{j}}(\bar{x})\right)^2 = \sum_{\bar{x}} \left(f(\bar{x})\psi_{\bar{j}}(\bar{x})\right)^2 + \sum_{\bar{x}} \sum_{\substack{\bar{x}' \\ \bar{x} \neq \bar{x}'}} \left(f(\bar{x})\psi_{\bar{j}}(\bar{x})\right)\left(f(\bar{x}')\psi_{\bar{j}}(\bar{x}')\right) \text{ (from (4}$$

Now, $\displaystyle\sum_{\bar{x}} \sum_{\substack{\bar{x}' \\ \bar{x} \neq \bar{x}'}} \psi_{\bar{j}}(\bar{x})\psi_{\bar{j}}(\bar{x}') = \sum_{\bar{x}} \sum_{\substack{\bar{x}' \\ \bar{x} \neq \bar{x}'}} (-1)^{\bar{j}.\bar{x}}(-1)^{\bar{j}.\bar{x}'} = \sum_{\bar{x}} \sum_{\substack{\bar{x}' \\ \bar{x} \neq \bar{x}'}} (-1)^{\bar{j}.\bar{x}+\bar{j}}$

If we fix $\bar{x}$, there are $2^n - 1$ different $\bar{x}'$ so that $\bar{x} \neq \bar{x}'$ and we have $2^n - 1$ tuples of $\bar{j}.\bar{x} + \bar{j}.\bar{x}'$, one for each different $\bar{x}'$.

If $\bar{j}.\bar{x}$ is odd, then arguing as (2), we have 2^{n-1} values of $\bar{x}'$ for which $\bar{j}.\bar{x}'$ is even and $2^{n-1} - 1$ values of $\bar{x}'$ for which $\bar{j}.\bar{x}'$ is odd $\Rightarrow$ we have 2^{n-1} values of $\bar{x}'$ for which $\bar{j}.\bar{x} + \bar{j}.\bar{x}'$ is odd and $2^{n-1} - 1$ values of $\bar{x}'$ for which $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is even.

Again, If $\bar{j}.\bar{x}$ is even, then arguing similarly, we have 2^{n-1} values of $\bar{x}'$ for which $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is even and $2^{n-1}-1$ values of $\bar{x}'$ for which $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is odd.

From (2), we know that for all $\bar{x}$ there are exactly 2^{n-1} values of $\bar{x}$ for which $\bar{j}.\bar{x}$ is odd and exactly 2^{n-1} values of $\bar{x}$ for which $\bar{j}.\bar{x}$ is even.

Hence, in the double summation,

the total number of cases where $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is odd = $2^{n-1}(2^{n-1})+2^{n-1}(2^{n-1}-1)=2^{n-1}(2^n-1)$

the total number of cases where $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is even = $2^{n-1}(2^{n-1}-1)+2^{n-1}(2^{n-1})=2^{n-1}(2^n-1)$

Hence, out of $2^n(2^n-1)$ tuples in the double summation (for each of 2^n choices for $\bar{x}$ there are exactly $2^{n-1}-1$ choices for $\bar{x}'$) for exactly half of them $\bar{j}.\bar{x}+\bar{j}.\bar{x}'$ is even, for the other half, it's odd.

$$\therefore \sum_{\bar{x}}\sum_{\substack{\bar{x}' \\ \bar{x}\neq\bar{x}'}} \psi_j(\bar{x})\psi_j(\bar{x}') = \sum_{\bar{x}}\sum_{\substack{\bar{x}' \\ \bar{x}\neq\bar{x}'}} (-1)^{\bar{j}.\bar{x}+\bar{j}.\bar{x}'}$$

$$= 2^{n-1}(2^n-1)(-1)^{\text{even number}} + 2^{n-1}(2^n-1)(-1)^{\text{odd number}}$$

$$= 2^{n-1}(2^n-1)(1-1)=0 \tag{5}$$

Also, $\sum_{\bar{x}}\left(f(\bar{x})\psi_j(\bar{x})\right)^2 = \sum_{\bar{x}} f^2(\bar{x})\psi_j{}^2(\bar{x}) = \sum_{\bar{x}} f^2(\bar{x})(-1)^{2(\bar{j}.\bar{x})} = \sum_{\bar{x}} f^2(\bar{x})$ (6)

Combining (5) and (6), we have,

$$\left(\sum_{\bar{x}} f(\bar{x})\psi_j(\bar{x})\right)^2 = \sum_{\bar{x}}\left(f(\bar{x})\psi_j(\bar{x})\right)^2 + \sum_{\bar{x}}\sum_{\substack{\bar{x}' \\ \bar{x}\neq\bar{x}'}} \left(f(\bar{x})\psi_j(\bar{x})\right)\left(f(\bar{x}')\psi_j(\bar{x}')\right)$$

$$= \sum_{\bar{x}} f^2(\bar{x}) + 0 = \sum_{\bar{x}} f^2(\bar{x}) \tag{7}$$

Combining (3) and (7), we have,

$$\sum_j w_j{}^2 = \sum_j\left(\frac{1}{2^n}\sum_{\bar{x}\in\{0,1\}^n} f(\bar{x})\psi_j(\bar{x})\right)^2 = \frac{1}{2^{2n}}\sum_j\left(\sum_{\bar{x}\in\{0,1\}^n} f(\bar{x})\psi_j(\bar{x})\right)^2$$

$$= \frac{1}{2^{2n}}\sum_j\sum_{\bar{x}} f^2(\bar{x}) = \frac{1}{2^{2n}}\sum_{\bar{x}} f^2(\bar{x})\sum_{j\in\{0,1\}^n} 1 = \frac{1}{2^{2n}}\left(\sum_{\bar{x}} f^2(\bar{x})\right).2^n$$

$$= \frac{1}{2^n}\left(\sum_{\bar{x}} f^2(\bar{x})\right) \quad (\text{Proved})$$

1. (b)

The frequency domain Fourier coefficients X_k are obtained from the spatial domain data x_n using the following equation using DFT (the same definition is used in Matlab's FFT):

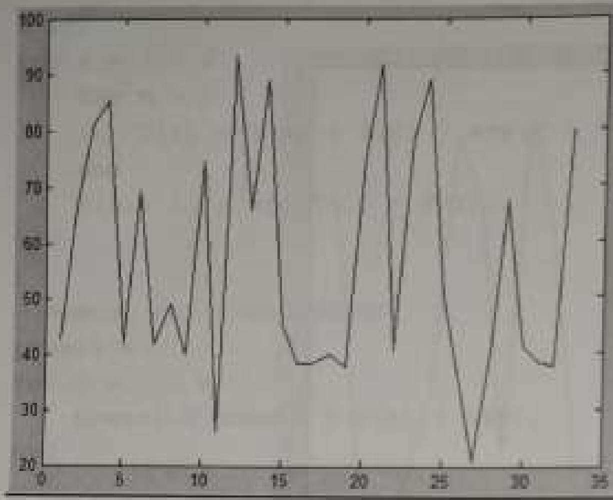
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn} \qquad k = 0, \ldots, N-1$$

```
% Time Series data already loaded in X
N = size(X, 1);
Y = zeros(N, 1);

% DFT
for k = 1 : N
    for n = 1 : N
        Y(k) = Y(k) + X(n) * exp((-2 * pi * i / N) * (k - 1) * (n - 1));
    end
end

% Matlab FFT
Z = fft(X);

% Plot
x = 1 : 1 : N;
plot(x, X, x, Y, x, Z);
legend('X', 'Y by DFT', 'Y by MatLab FFT');
```

Both Y and Z give the same result, the Fourier coefficients are (from 0^{th} to $(N-1)^{th}$):

Matlab Output

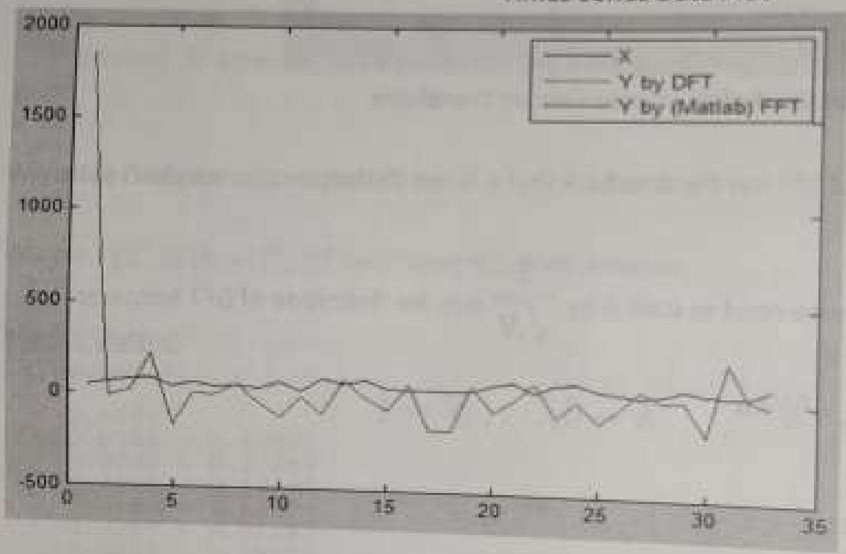
```
1.0e+003 *

   1.8525
  -0.0168 - 0.0557i
   0.0259 - 0.1057i
   0.2190 - 0.0482i
  -0.1655 - 0.0430i
   0.0084 - 0.0223i
  -0.0007 - 0.1040i
   0.0571 + 0.0714i
  -0.0330 + 0.1363i
  -0.1143 - 0.0413i
  -0.0079 + 0.1147i
  -0.0922 + 0.0761i
   0.0866 - 0.0407i
  -0.0034 + 0.0288i
  -0.0719 + 0.0686i
   0.0629 - 0.0447i
  -0.1752 - 0.1053i
  -0.1752 + 0.1053i
   0.0629 + 0.0447i
  -0.0719 - 0.0686i
  -0.0034 - 0.0288i
   0.0866 + 0.0407i
  -0.0922 - 0.0761i
  -0.0079 - 0.1147i
  -0.1143 + 0.0413i
  -0.0330 - 0.1363i
   0.0571 - 0.0714i
  -0.0007 + 0.1040i
   0.0084 + 0.0223i
  -0.1655 + 0.0430i
   0.2190 + 0.0482i
   0.0259 + 0.1057i
  -0.0168 + 0.0557i
```



Time domain plot

Frequency domain plot

Times Series Data Plot

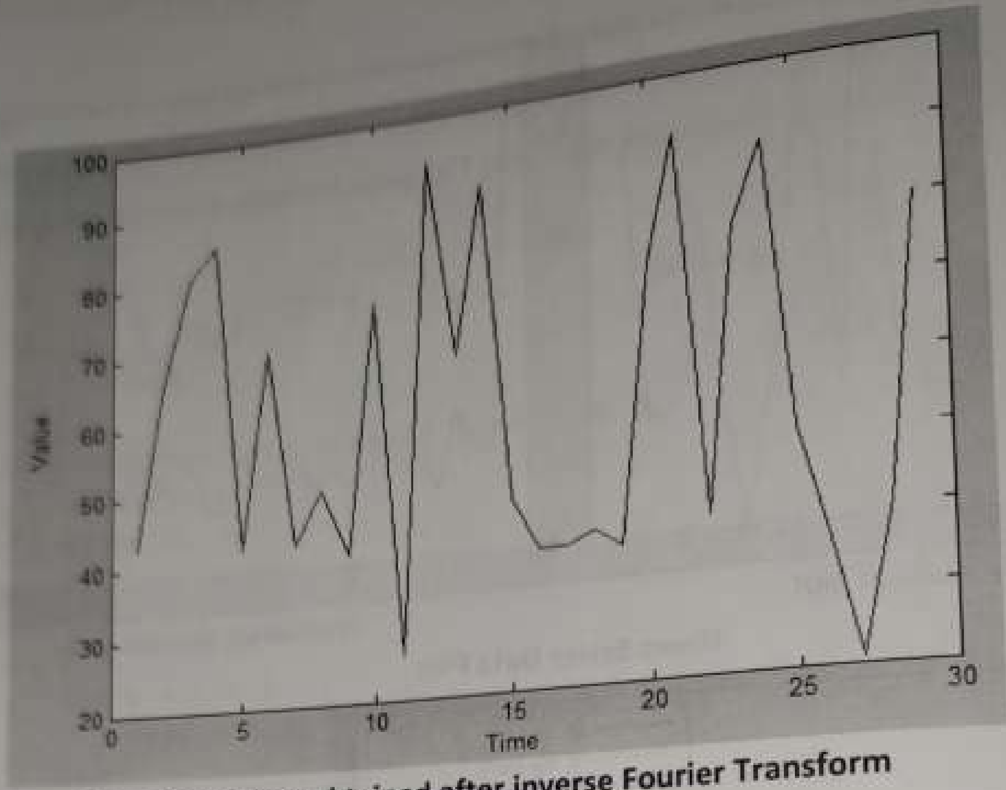


The inverse DFT can be expressed by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \qquad n = 0, \ldots, N-1$$

If we do the IDFT, we get back the same time series data.

```
% IDFT
for n = 1 : N
    X(n) = 0;
    for k = 1 : N
        X(n) = X(n) + Y(k) * exp((2 * pi * i / N) * (k - 1) * (n - 1));
    end
    X(n) = (1 / N) * X(n);
end
% Matlab IFFT
X = ifft(Z);
```



Time Series data obtained after inverse Fourier Transform

But the above definition of DFT has the drawback that it is **not unitary** and hence don't satisfy Parseval's theorem.

In order to make it unitary we need to scale it by $\frac{1}{\sqrt{N}}$ and the definition of DFT becomes:

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}, \qquad k = 0, \dots, N-1$$

Accordingly, IDFT becomes:

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn}, \qquad n = 0, \dots, N-1$$

and we see that it satisfies Parseval's theorem:

$$\sum_{k=0}^{N-1} X_k^{\ 2} = \sum_{n=0}^{N-1} x_n^{\ 2}$$

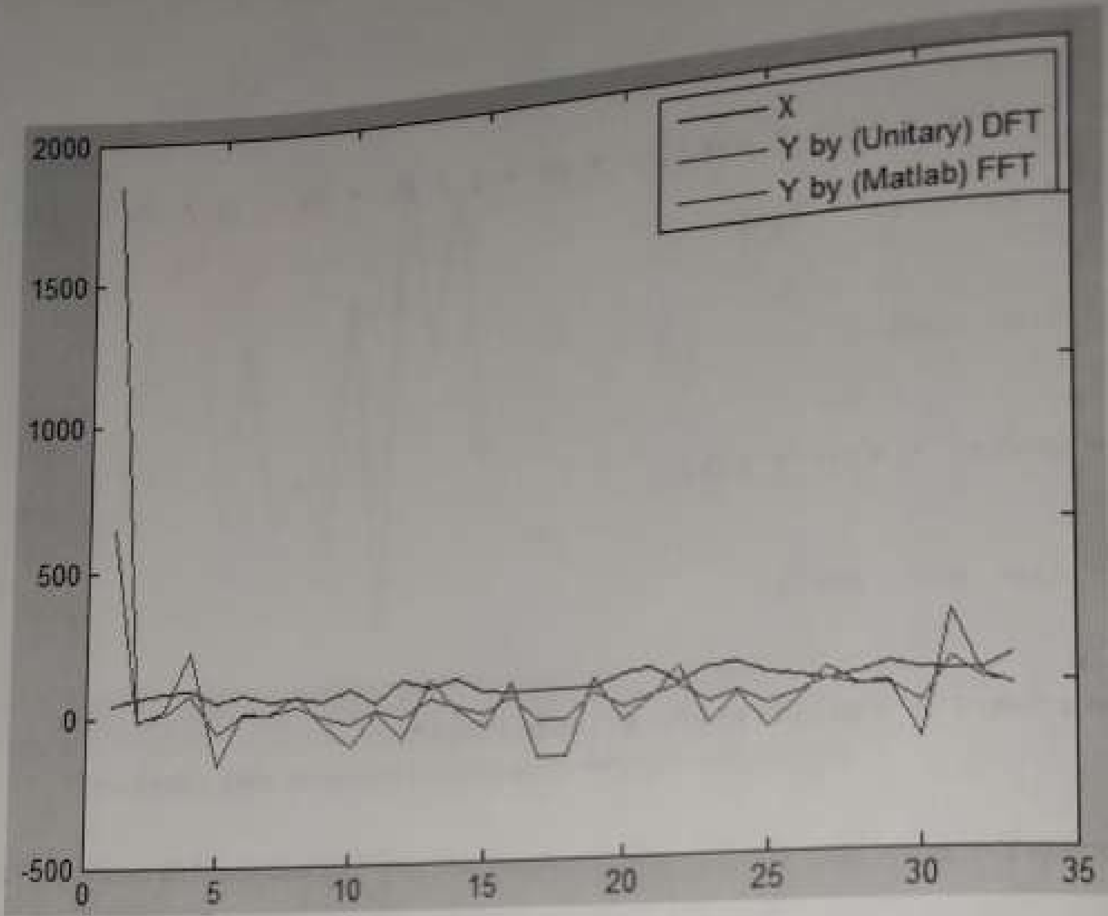
```
% DFT
for k = 1 : N
    for n = 1 : N
        Y(k) = Y(k) + X(n) * exp((-2 * pi * i / N) * (k - 1) * (n - 1));
    end
    Y(k)= (1 / sqrt(N)) * Y(k);
end

% energy for time domain
power1 = 0;
for n = 1 : N
    power1 = power1 + X(n) * X(n);
end

% energy for frequency domain
power2 = 0;
for k = 1 : N
    power2 = power2 + real(Y(k))^2 + imag(Y(k))^2;
end
```

We verify that power1 ≈ power2.

We get Y (0^{th} to $(N-1)^{th}$ DFT coefficients) scaled down to:

Matlab Output

```
1.0e+002 *

   6.4497
  -0.0586 - 0.1939i
   0.0902 - 0.3681i
   0.7625 - 0.1679i
  -0.5763 - 0.1497i
   0.0294 - 0.0778i
  -0.0024 - 0.3621i
   0.1987 + 0.2487i
  -0.1149 + 0.4746i
  -0.3979 - 0.1437i
  -0.0274 + 0.3994i
  -0.3208 + 0.2649i
   0.3016 - 0.1418i
  -0.0119 + 0.1001i
  -0.2504 + 0.2389i
   0.2189 - 0.1555i
  -0.6098 - 0.3665i
  -0.6098 + 0.3665i
   0.2189 + 0.1555i
  -0.2504 - 0.2389i
  -0.0119 - 0.1001i
   0.3016 + 0.1418i
  -0.3208 - 0.2649i
  -0.0274 - 0.3994i
  -0.3979 + 0.1437i
  -0.1149 - 0.4746i
   0.1987 - 0.2487i
  -0.0024 + 0.3621i
   0.0294 + 0.0778i
  -0.5763 + 0.1497i
   0.7625 + 0.1679i
   0.0902 + 0.3681i
  -0.0586 + 0.1939i
```



In this case, power1 = power2 = 1.0965e+005 and **Parseval's theorem is satisfied**.

(c) To find the % energy preserved when we choose only the first m Fourier coefficients (and ignore the rest of the higher order coefficients), we use the following formula:

$$p = \frac{\sum_{k=0}^{m-1} X_k^{\ 2}}{\sum_{k=0}^{N-1} X_k^{\ 2}} \times 100 = \frac{\sum_{k=0}^{m-1} X_k^{\ 2}}{\sum_{n=0}^{N-1} x_n^{\ 2}} \times 100$$

(by Parseval's theorem denominators are equal)

By condition, $p \geq 80 \Rightarrow$ we have to solve the following inequality for m:

$$\frac{\sum_{k=0}^{m-1} X_k^{\ 2}}{\sum_{n=0}^{N-1} x_n^{\ 2}} \geq 0.8$$

where m being the only unknown for the above equation.

The following Matlab code can be used to serve our purpose:

```
% energy for time domain
power1 = 0;
for n = 1 : N
    power1 = power1 + X(n) * X(n);
end

% energy for frequency domain
power2 = 0;
display('#Fourier coefficients  %Energy preserved');
for k = 1 : N
    power2 = power2 + real(Y(k))^2 + imag(Y(k))^2;
    fprintf('          %d                    %f \n', k, 100 * (power2 / power1));
end
```

The output produced by the above code:

Matlab Output

```
#Fourier coefficients   %Energy preserved
          1                  87.548549
          2                  87.524472
          3                  87.116794
          4                  87.742105
          5                  88.757123
          6                  88.736588
          7                  88.464337
          8                  88.625224
          9                  87.949525
         10                  88.479975
         11                  88.099773
         12                  87.811103
         13                  87.780148
         14                  87.754329
         15                  87.514476
         16                  87.421210
         17                  88.861897
         18                  88.421231
         19                  88.614507
         20                  88.878226
         21                  88.862425
         22                  89.191652
         23                  89.618361
         24                  89.330206
         25                  89.379345
         26                  89.162723
         27                  88.907693
         28                  88.628060
         29                  88.626747
         30                  88.915558
         31                  90.618513
         32                  90.490220
         33                  90.370469
```

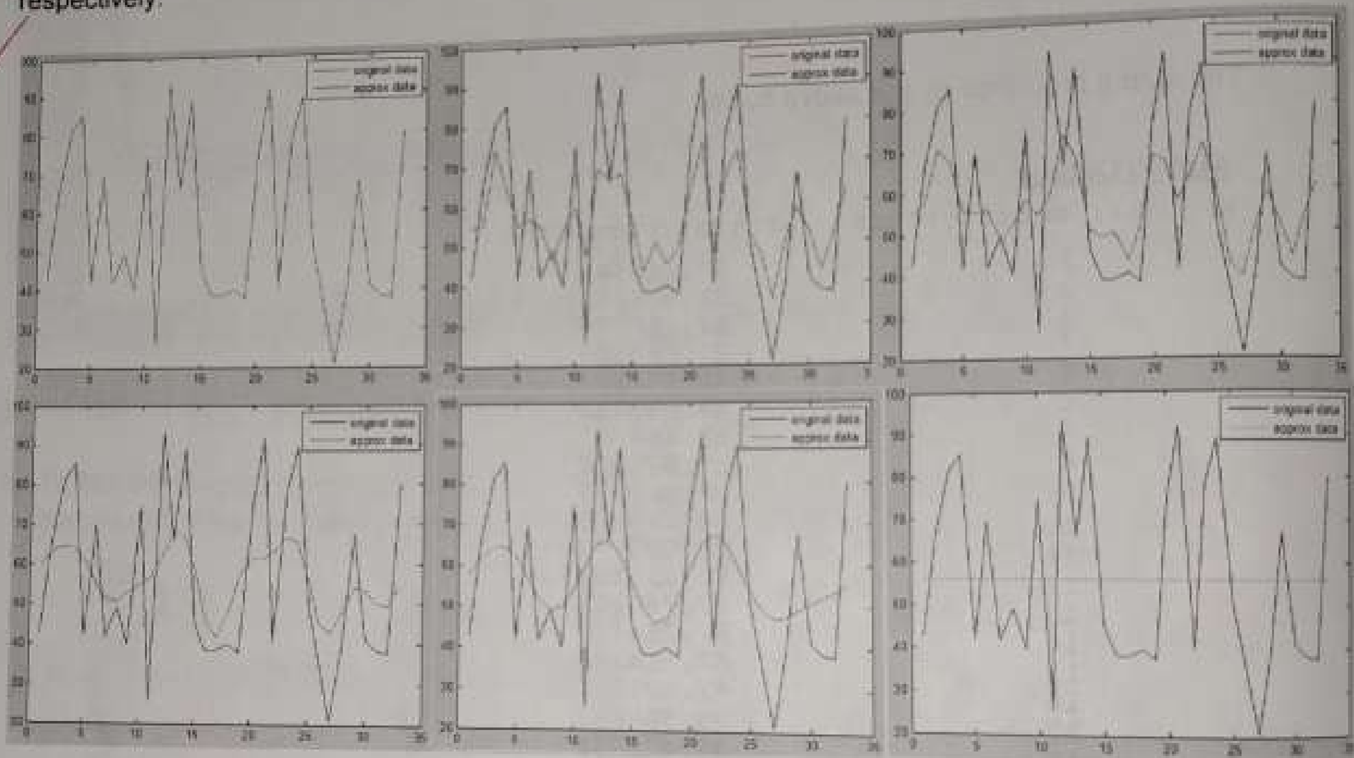
As seen from the above output, if we use the 1st Fourier Coefficient only (i.e., the 0th coefficient) and ignore the rest, then more than 80% (more precisely around 87.5%) of overall energy is preserved, i.e., it can be represented by the Fourier coefficient (signature).

$$X_0 = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x_n e^{-0} = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x_n$$

thereby preserving more than 80% of energy.

This is the Fourier Signature of the image in the sense that if we do IDFT using this coefficient only we can represent the original data approximately with high degree of accuracy.

The following figure shows how the lower Fourier signatures can approximate the original data, if we retain all, half, 1/3rd, 1/4th, square root # of original coefficients and finally just 1 (0^{th}) coefficient respectively.




```
% approximation using (unitary) IDFT
for n = 1 : N
    X(n) = 0;
    % use first m Fourier coefficients to approximate the data
    % m = 1, ..., N
    for k = 1 : m
        X(n) = X(n) + Y(k) * exp((2 * pi * i / N) * (k - 1) * (n - 1));
    end
    X(n) = (1 / sqrt(N)) * X(n);
end
```

2. Since PCA is a random projection technique, we do PCA on the Iris data and then take projection.

```
% the last 4 columns of Iris data is already loaded into X
% find # of row vectors (data tuples)
n = size(X, 1);
% normalize X
X = zscore(X);
% do PCA to obtain the set of orthogonal eigen vectors V
[V, S] = princomp(X);
% choose random projection matrix: two most dominant eigen vectors of V
V = V(:, 1:2);
% project X along V
Y = X * V;
% find the inner product error matrix
E = Y * Y' - X * X';
% average error in inner product
m = mean(mean(E));
% variance of error in inner product
v = var(var(E));
```

With the following output mean and variance, respectively, both are very small, as expected.

```
m =

  1.0329e-017

v =

    0.0010
```

We can use **random projection matrix** instead, by generating a 4 x 2 random projection matrix, with elements (*i.i.d.* random variables) from a normal distribution with mean 0 and variance 1, as below:

```
% the last 4 columns of Iris data is already loaded into X
% find # of row vectors (data tuples)
n = size(X, 1);
% normalize X
X = zscore(X);
% the last 4 columns of Iris data is already loaded into X
% generate a random 4 x 2 (projection) matrix by drawing samples (pseudo) randomly from
% a Gaussian population with mean 0 and variance 1
% hence the sample random variables can be thought of as i.i.d. variables.
V = randn(4, 2);
% project X along V
Y = X * V;
% find the inner product error matrix
E = Y * Y' - X * X';
% average error in inner product
m = mean(mean(E));
% variance of error in inner product
v = var(var(E));
```

With the following output:

```
m =

   1.6185e-018

v =

    4.8037
```

Again with almost zero mean and low variance of error in between inner-product matrices, i.e., the random projection is preserving the inner products.

The above two techniques requires to use z-score normalization on the data (mean adjusted) as starting point. There is another random projection technique, which is **SVD** that does not require this initial normalization.

```
% the last 4 columns of Iris data is already loaded into X
% find # of row vectors (data tuples)
n = size(X, 1);
% do SVD to get the random projection matrix (orthogonal)
[U,S,V] = svd(X);
% project X along V
Y = X * V;
% find the inner product error matrix
E = Y * Y' - X * X';
% average error in inner product
m = mean(mean(E));
% variance of error in inner product
v = var(var(E));
```

With the following outputs:

```
m =

   2.0276e-012

v =

   2.1902e-049
```

Again the mean and variance are very low, approaching zero, i.e., the random projection is preserving the inner products.