<u>**FINAL REPORT**</u>

BEDANTA GAUTOM

RA2011033010048

Github Link –

https://github.com/BedantaGautom/IIT_CSE_Summer_Internship

05/06/2023 – 06/06/2023 :

- Speech Production :
  - Conceptualization – what to say.
  - Formulation – takes conceptual entities as input and connect it with releavant words to bulid syntatics, morphological and phonological structure.
  - Articulation – structure is phonetically encoded and articulated, resulting in speech.
- Speech Processing Models :
  - Dell Model
  - LRM Model
  - LEWISS Model
- Waveform VS Spectrogram :
  - Waveform – displays change in amplitude over time.
  - Spectrogram - displays change in frequency over time.
- ZCR : Zero Crossing Rate – rate at which signal changes from +ve to 0 to –ve or vice versa.
- LPC : Linear Predictive Coding – reducing amount of information needed to represent speech signal.
- MFCC : Mel-frequency cepstral Coefficients – technique to extract the featuires from the audio signal.
- .wav to Spectrogram in algorithm :
  - Load .wav file.
  - Compute spectrogram with consecutive Fourier transforms using spectrogram() method.
  - Create a pseudocolor plot with non-rectangular grid using pcolormesh() method.
  - To display, use inshow() method with spectrogram than show() method.
- Spectrogram to .wav algorithm :
  - To rebulid wav, convert your magnitude spectrum to Fast Fourier Transform with zero plane.
  - Rebuild an excitaiton from the pitch extracted from original signal.
  - Convolve that excitaiton with the Fast Fourier Transform.

07/06/2023 – 09/06/2023 :

- Frechet Distance Algorithm :
  1. Given two curve or paths, eah represented by sequence of points, call it curve A and curve B.
  2. Calculate Euclidean distance between eact point from curve A and curve B. This create a distance matrix.
  3. Initialize memorization table to store intermediate result for the algorithm.
  4. Define recursive function that calculates Frechet distance between two subcurves, one from curve A and other from curve B.
  5. Input - Indices of start and end point of curves.
  6. Check if result for given indices already present in memorization table.
- To prepare slides for Frechet Distance algorithm and Clustering algorithm.
  - Link – https://drive.google.com/drive/folders/1cBxGAPtaq6uyec0sQ5-2pkwf47uxJCd8?usp=sharing

12/06/2023 :

- Distance Metrics, Applications and its uses :
  - Hamming Distance – It measures similarity between two strings of same length. It calculates distance between two binary vectors / binary strings or bitstrings.
    - Uses – Error detection/correction, Coding theory, Comparing equal length data words.
    - Hamming Distance = (sum for i to N abs(v1[i] – v2[i])/N

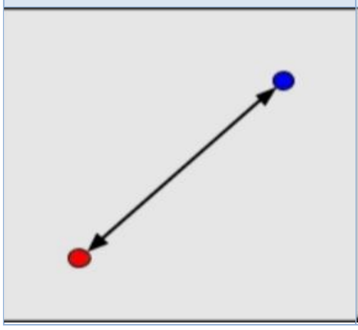
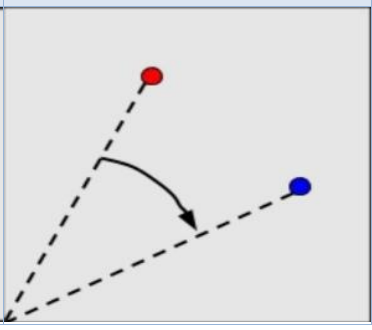
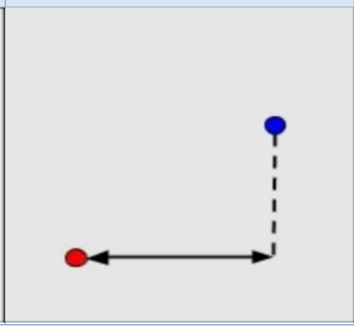
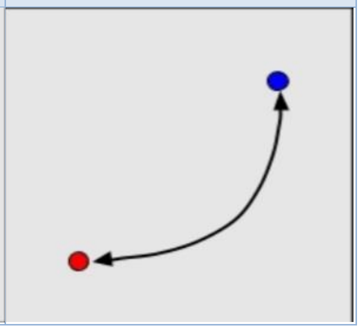
  - Euclidean Distance – It calculates shortest distance between two vectors. It calculates distance between two rows of data having numerical values. It measures straight line distance.
    - Uses – Molecular Conformation, Localization of server networks and staistics.
    - Euclidean Distance = sqrt(sum for i to N (v1[i] – v2[i])^2)

  - Manhattan Distance – It calculates sum of absolute difference between points across all dimensions.
    - Uses – Regression Analysis, Frequency Distribution, Compressed Sensing
    - Manhattan Distance = sum for i to N sum|v1[i] – v2[i]|

  - Minkowski Distance – It is generalized form of Euclidean distance and Manhattan Distance.
    - Uses – Fuzzy Clustering
    - Minkowski Distance = (sum for i to N (abs(v1[i] – v2[i])^p)^(1/p)
      Where p= order.
      P = 1, Manhattan Distance
      P = 2, Euclidean Distance

14/06/2023 :

- Comparision Analysis :
  - Euclidean Distance
  - Cosine Similarity
  - Chevyshev Distance
  - Minkowski Distance

Eucledian Distance	Cosine Similarity	Chebyshev Distance	Minkowski Distance
Calculaltes straight line distance between two points in Eucledian space.	Calculate similarity between two or more vectors.	Calculates greatest of difference between two vectors along any coordiante dimension.	It is generalized form of Eucledian and Manhattan distance which can be adjusted based on order parameter p.
$D = sqrt((x_1-y_1)^2 + (x_2-y_2)^2 +....+ (x_n-y_n)^2)$	$D = sqrt((x_1-y_1)^2 + (x_2-y_2)^2 +....+ (x_n-y_n)^2)$	$D = max(\|x_1-y_1\| + \|x_2-y_2\| +....+ \|x_n-y_n\|)$	$D=(\|x_1-y_1\|^p + \|x_2-y_2\|^p +....+ \|x_n-y_n\|^p)^{(1/p)}$
Applications – • Molecular Conformation • Localization of sensor networks and statistics.	Applications – • Data Mining • Recommendation system in machine learning. • Computer Vision	Applications – • Clustering algorithm like K-Means or DBSCAN. • Time Series Analysis • Image Processing	Applications – • Fuzzy Clustering • Time Seies Analysis
			

- Implementation in C :
  - Link – https://github.com/BedantaGautom/IIT_CSE_Summer_Internship/tree/main/distance_algorithms

16/06/2023 :

- Frechet Distance Pseudo-Code and Dry Run :
  - Pseudo- Code –

Date :

→ Frechet distance pseudocode:

```
// function to calculate Eucledian distance
Function eucledian (point p1, point p2):
    return sqrt((p1.x - p2.x)^2 + (p1.y - p2.y)^2)

p_length = P.size()      // Number of rows in distance matrix
q_length = Q.size()      // No. of columns in distance matrix

P = { }                  // Row values are stored
Q = { }                  // Column values are stored

distance_matrix (p_length,(q_length, -1));

// Store initial value

distance-matrix [0][0] = eucledian (P[0], Q[0])

for i = 1 to p_length do:

distance_matrix [i][0] = max (distance_matrix [i-1][0]
                         eucledian (P[i], Q[0]));
// Calculate distance for remaining cells
for j = 1 to q_length do:

distance_matrix [0] [j] = max (distance_matrix
[0] [j-1], eucledian (P[0], Q[j]));
```

Date :

for i=1 to p_length do:
    for j=1 to q_length do:

distance_matrix [i] [j] = max(min({distance_matrix [i-1][j], distance_matrix [i][j-1], distance_matrix [i-1][j-1]}), euclediean (P[i], Q[j]))

// Returns frechet distance
return distance_matrix [p_length-1][q_length-1]

- Dry Run –

Date :

$P = \{\{2,1\}, \{3,1\}\}$

$Q = \{\{2,0\}, \{3,0\}, \{4,0\}\}$

Eucledian $(p1, p2) = sqrt[(2-3)^2 + (1-1)^2]$

$= sqrt[1] = 1$

Eucledian $(p1, q1) = sqrt[(2-2)^2 + (1-0)^2]$

$= 1$

Eucledian $(p1, q_2) = sqrt[(2-3)^2 + (1-0)^2]$

$= sqrt[1+1]$

$= \sqrt{2} = 1.414 \simeq 1.4$

Eucledian $(p1, q3) = sqrt[(2-4)^2 + (1-0)^2]$

$= sqrt[2^2+1]$

$= sqrt[5] = 2.23 \simeq 2.3$

Eucledian $(p_2, q_1) = sqrt[(3-2)^2 + (1-0)^2]$

$= sqrt[2]$

$\simeq 1.4$

Eucledian $(p_2, q_2) = sqrt[(3-3)^2 + (1-0)^2]$

$= 1$

Eucledian $(p2, q3) = sqrt[(3-4)^2 + (1-0)^2]$

$= sqrt[2] \simeq 1.4$

Date :

→ Distance Matrix

	q1	q2	q3
p1	1	1.4	2.3
p2	1.4	1	1.4

Now,

$\min[(p_1, q_1), (p_1, q_2), (p_1, q_3)] = \min[1, 1.4, 2.3]$
$= 1$

$\min[(p_2, q_1), (p_2, q_2), (p_2, q_3)] = \min[1.4, 1, 1.4]$
$= 1$

$\max(p_1, p_2) = (1, 1) = 1$

Again,

$\min[(q_1, p_1), (q_1, p_2)] = \min[1, 1.4] = 1$

$\min[(q_2, p_1), (q_2, p_2)] = \min[1.4, 1] = 1$

$\min[(q_3, p_1), (q_3, p_2)] = \min[2.3, 1.4] = 1.4$

Date :

$\max(q_1, q_2, q_3) = (1, 1, 1.4) = 1.4$

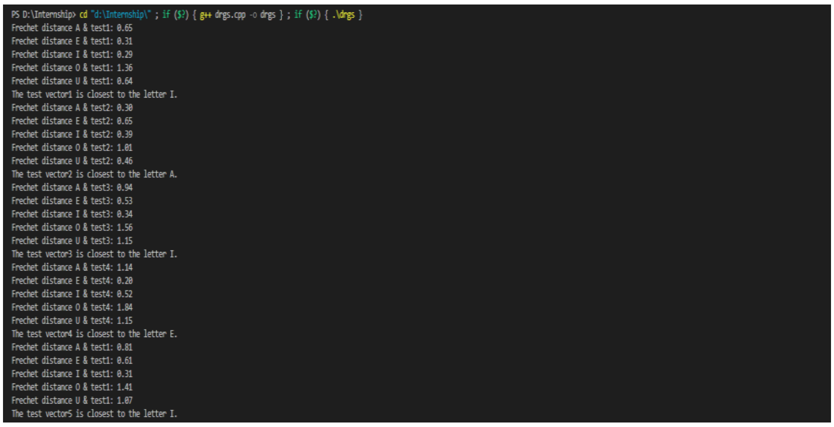
$\max[(P), (Q)] = (1, 1.4)$
$= 1.4$

$\therefore$ Frechet distance $= 1.4$

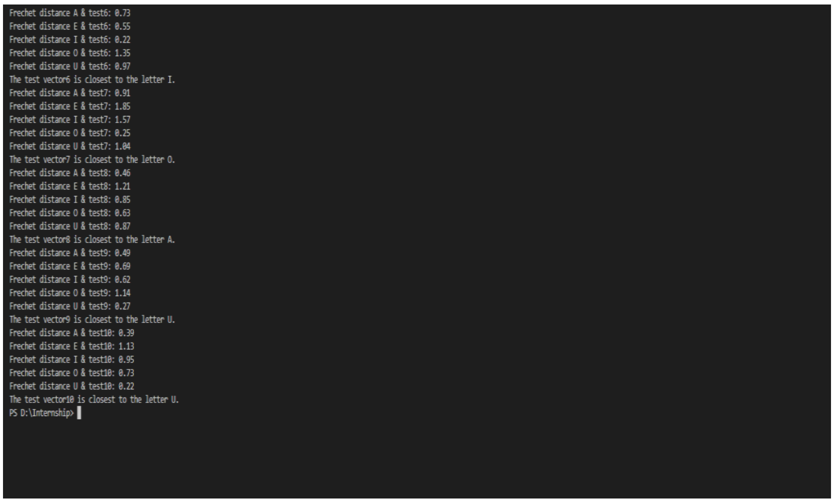
Duke
SAVE ENERGY
PUMPS & PIPES

19/06/2023 :

- Frechet Distance Implementation in C for reference vectors and test vectors :
  - Link- https://github.com/BedantaGautom/IIT_CSE_Summer_Internship/blob/main/frechet.cpp
  - Output –



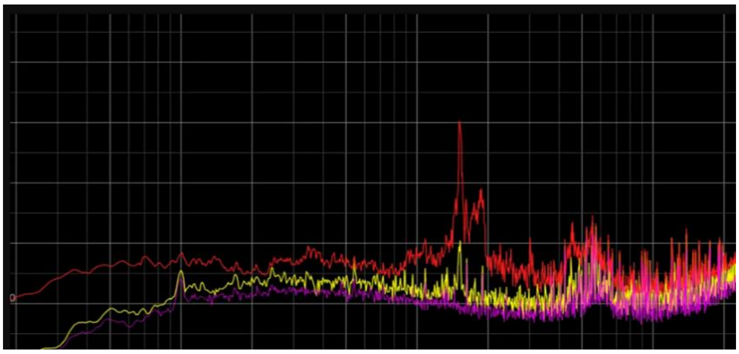
```
PS D:\Internship> cd "d:\Internship\" ; if ($?) { g++ drgs.cpp -o drgs } ; if ($?) { .\drgs }
Frechet distance A & test1: 0.65
Frechet distance E & test1: 0.31
Frechet distance I & test1: 0.29
Frechet distance O & test1: 1.36
Frechet distance U & test1: 0.64
The test vector1 is closest to the letter I.
Frechet distance A & test2: 0.30
Frechet distance E & test2: 0.65
Frechet distance I & test2: 0.39
Frechet distance O & test2: 1.01
Frechet distance U & test2: 0.46
The test vector2 is closest to the letter A.
Frechet distance A & test3: 0.94
Frechet distance E & test3: 0.53
Frechet distance I & test3: 0.34
Frechet distance O & test3: 1.56
Frechet distance U & test3: 1.15
The test vector3 is closest to the letter I.
Frechet distance A & test4: 1.14
Frechet distance E & test4: 0.20
Frechet distance I & test4: 0.52
Frechet distance O & test4: 1.84
Frechet distance U & test4: 1.15
The test vector4 is closest to the letter E.
Frechet distance A & test1: 0.81
Frechet distance E & test1: 0.61
Frechet distance I & test1: 0.31
Frechet distance O & test1: 1.41
Frechet distance U & test1: 1.07
The test vector5 is closest to the letter I.
```



```
Frechet distance A & test6: 0.73
Frechet distance E & test6: 0.55
Frechet distance I & test6: 0.22
Frechet distance O & test6: 1.35
Frechet distance U & test6: 0.97
The test vector6 is closest to the letter I.
Frechet distance A & test7: 0.91
Frechet distance E & test7: 1.85
Frechet distance I & test7: 1.57
Frechet distance O & test7: 0.25
Frechet distance U & test7: 1.04
The test vector7 is closest to the letter O.
Frechet distance A & test8: 0.46
Frechet distance E & test8: 1.21
Frechet distance I & test8: 0.85
Frechet distance O & test8: 0.63
Frechet distance U & test8: 0.87
The test vector8 is closest to the letter A.
Frechet distance A & test9: 0.49
Frechet distance E & test9: 0.69
Frechet distance I & test9: 0.62
Frechet distance O & test9: 1.14
Frechet distance U & test9: 0.27
The test vector9 is closest to the letter U.
Frechet distance A & test10: 0.39
Frechet distance E & test10: 1.13
Frechet distance I & test10: 0.95
Frechet distance O & test10: 0.73
Frechet distance U & test10: 0.22
The test vector10 is closest to the letter U.
PS D:\Internship>
```

- Dynamic Time Warping :
  - Dynamic Time Warping is used to compare the similarity or calculate the distance between time series with different length.
  - In time series analysis, dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. DTW has been applied to temporal sequences of video, audio, and graphics data — indeed, any data that can be turned into a linear sequence can be analysed with DTW.
  - The idea to compare arrays with different length is to build one-to-many and many-to-one matches so that the total distance can be minimised between the two.
  - Use Cases –
    - Sound Pattern Recognition
      - One use case is to detect the sound pattern of the same kind. Suppose we want to recognise the voice of a person by analysing his sound track, and we are able to collect his sound track of saying Hello in one scenario. However, people speak in the same word in different ways, what if he

speaks hello in a much slower pace like Heeeeeeelloooooo , we will need an algorithm to match up the sound track of different lengths and be able to identify they come from the same person.



- Stock Market
  - In a stock market, people always hope to be able to predict the future, however using general machine learning algorithms can be exhaustive, as most prediction task requires test and training set to have the same dimension of features. However, if you ever speculate in the stock market, you will know that even the same pattern of a stock can have very different length reflection on klines and indicators.

20/06/2023 :

- Dynamic Time Warping Pseudo-code :
    - Pseudo-code –

```
function dynamicTimeWarping(sequenceA, sequenceB):
    // Step 1: Define the input sequences
    lengthA = length(sequenceA)
    lengthB = length(sequenceB)

    // Step 2: Define a distance or similarity measure

    // Step 3: Create a cost matrix
    costMatrix = createMatrix(lengthA, lengthB)

    // Step 4: Initialize the cost matrix
    costMatrix[0][0] = distance(sequenceA[0], sequenceB[0])

    for i = 1 to lengthA:
        costMatrix[i][0] = costMatrix[i-1][0] + distance(sequenceA[i], sequenceB[0])

    for j = 1 to lengthB:
        costMatrix[0][j] = costMatrix[0][j-1] + distance(sequenceA[0], sequenceB[j])

    // Step 5: Calculate the cumulative cost
    for i = 1 to lengthA:
        for j = 1 to lengthB:
            cost = distance(sequenceA[i], sequenceB[j])
            minCost = min(costMatrix[i-1][j], costMatrix[i][j-1], costMatrix[i-1][j-1])
            costMatrix[i][j] = cost + minCost

    // Step 6: Find the optimal path
    path = []
    i = lengthA
    j = lengthB

    while i > 0 and j > 0:
        path.append((i, j))
        if i == 1 and j == 1:
            break
        if i > 1 and j > 1:
            minCost = min(costMatrix[i-1][j], costMatrix[i][j-1], costMatrix[i-1][j-1])
            if minCost == costMatrix[i-1][j-1]:
                i = i - 1
                j = j - 1
            elif minCost == costMatrix[i][j-1]:
                j = j - 1
            else:
                i = i - 1
```

```
        else if i > 1:
            i = i - 1
        else:
            j = j - 1

    // Step 7: Compute the DTW distance
    dtwDistance = costMatrix[lengthA][lengthB]

    return dtwDistance, path
```

- Implementation Link – https://github.com/BedantaGautom/IIT_CSE_Summer_Internship/blob/main/dtw.c

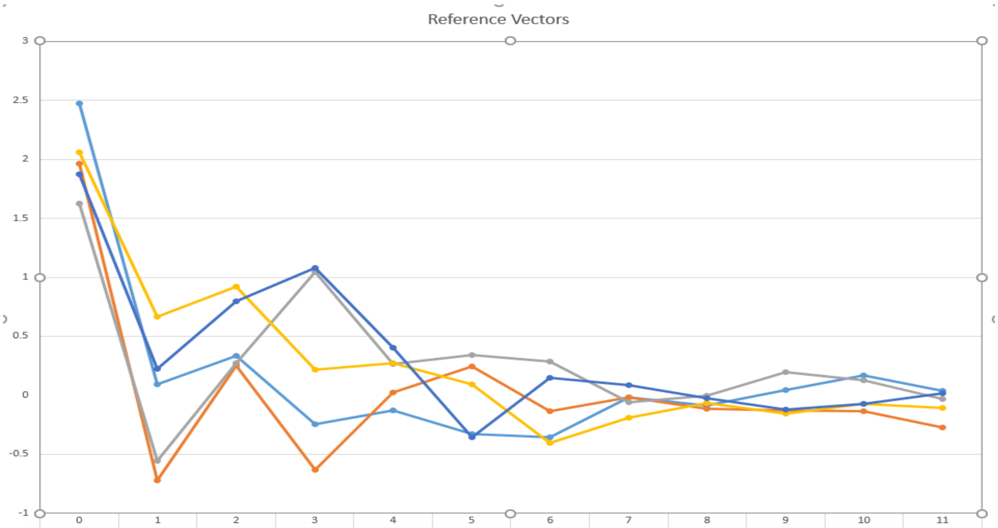
- Output –

```
PS D:\Internship> cd "d:\Internship\" ; if ($?) { gcc dtw1.c -o dtw1 } ; if ($?) { .\dtw1 }
DTW distance A & test1: 1.25
DTW distance E & test1: 5.02
DTW distance I & test1: 5.45
DTW distance O & test1: 6.99
DTW distance U & test1: 6.30
The test vector1 is closest to the letter A.
DTW distance A & test2: 1.86
DTW distance E & test2: 2.64
DTW distance I & test2: 4.03
DTW distance O & test2: 5.35
DTW distance U & test2: 5.20
The test vector2 is closest to the letter A.
DTW distance A & test3: 1.84
DTW distance E & test3: 1.29
DTW distance I & test3: 2.57
DTW distance O & test3: 5.33
DTW distance U & test3: 4.91
The test vector3 is closest to the letter E.
DTW distance A & test4: 1.84
DTW distance E & test4: 0.84
DTW distance I & test4: 2.88
DTW distance O & test4: 5.33
DTW distance U & test4: 4.45
The test vector4 is closest to the letter E.
DTW distance A & test5: 1.84
DTW distance E & test5: 3.16
DTW distance I & test5: 1.19
DTW distance O & test5: 5.33
DTW distance U & test5: 4.00
The test vector5 is closest to the letter I.
```

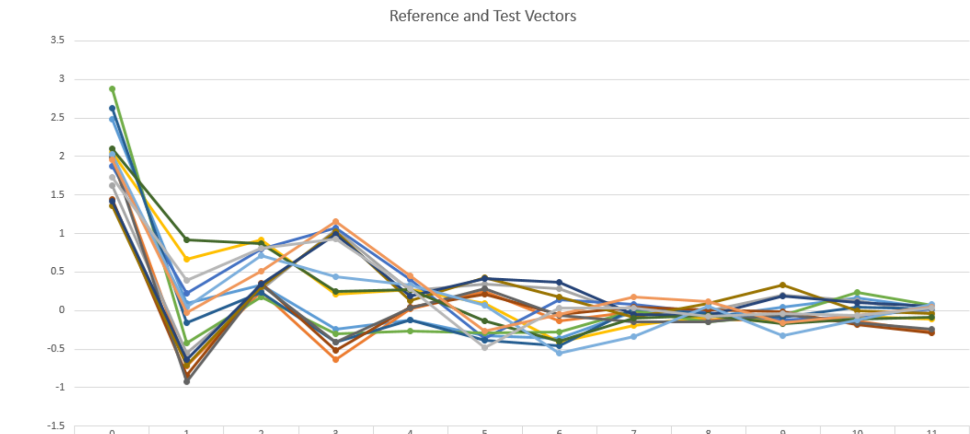
```
DTW distance A & test6: 1.84
DTW distance E & test6: 3.25
DTW distance I & test6: 0.82
DTW distance O & test6: 5.33
DTW distance U & test6: 3.41
The test vector6 is closest to the letter I.
DTW distance A & test7: 1.74
DTW distance E & test7: 4.29
DTW distance I & test7: 4.59
DTW distance O & test7: 5.51
DTW distance U & test7: 4.70
The test vector7 is closest to the letter A.
DTW distance A & test8: 1.84
DTW distance E & test8: 2.92
DTW distance I & test8: 2.13
DTW distance O & test8: 3.62
DTW distance U & test8: 1.51
The test vector8 is closest to the letter U.
DTW distance A & test9: 1.84
DTW distance E & test9: 3.48
DTW distance I & test9: 2.43
DTW distance O & test9: 1.69
DTW distance U & test9: 1.02
The test vector9 is closest to the letter U.
DTW distance A & test10: 1.87
DTW distance E & test10: 2.72
DTW distance I & test10: 3.23
DTW distance O & test10: 4.85
DTW distance U & test10: 3.77
The test vector10 is closest to the letter A.
PS D:\Internship>
```

23/06/2023 :

- Reference vectors (A,E,I,O,U) and test vectors graph :



	0	1	2	3	4	5	6	7	8	9	10	11
A	2.477543	0.091351	0.330765	-0.247886	-0.126224	-0.331168	-0.359815	-0.01934	-0.087245	0.04077	0.164893	0.037463
E	1.965155	-0.720578	0.251746	-0.636152	0.025966	0.243731	-0.133845	-0.019735	-0.117502	-0.126781	-0.137189	-0.273721
I	1.625304	-0.557864	0.272417	1.046378	0.26214	0.342437	0.288025	-0.058243	-0.005909	0.198275	0.12624	-0.033651
O	2.057388	0.667621	0.918405	0.215574	0.274346	0.090676	-0.404036	-0.190703	-0.069546	-0.155212	-0.073438	-0.108692
U	1.877386	0.224637	0.796147	1.077534	0.399793	-0.35618	0.145718	0.085206	-0.025559	-0.124376	-0.070185	0.018517



	0	1	2	3	4	5	6	7	8	9	10	11
Series1	2.477543	0.091351	0.330765	-0.247886	-0.126224	-0.331168	-0.359815	-0.01934	-0.087245	0.04077	0.164893	0.037463
Series2	1.965155	-0.720578	0.251746	-0.636152	0.025966	0.243731	-0.133845	-0.019735	-0.117502	-0.126781	-0.137189	-0.273721
Series3	1.625304	-0.557864	0.272417	1.046378	0.26214	0.342437	0.288025	-0.058243	-0.005909	0.198275	0.12624	-0.033651
Series4	2.057388	0.667621	0.918405	0.215574	0.274346	0.090676	-0.404036	-0.190703	-0.069546	-0.155212	-0.073438	-0.108692
Series5	1.877386	0.224637	0.796147	1.077534	0.399793	-0.35618	0.145718	0.085206	-0.025559	-0.124376	-0.070185	0.018517
Series6	2.88164	-0.418731	0.180176	-0.300379	-0.261485	-0.294065	-0.283541	0.001975	-0.142601	-0.05938	0.239747	0.069706
Series7	2.630109	-0.163863	0.233121	-0.41098	-0.126481	-0.384984	-0.455946	0.024326	-0.03733	-0.083877	0.042574	0.020905
Series8	1.445283	-0.843818	0.35739	-0.516838	0.038891	0.209207	-0.049682	0.041211	0.003941	-0.014337	-0.188222	-0.289964
Series9	1.991529	-0.920823	0.331697	-0.41363	0.035581	0.279182	-0.068747	-0.149884	-0.143787	-0.056978	-0.163985	-0.242462
Series10	1.35934	-0.714635	0.309969	1.025957	0.133121	0.42644	0.181778	-0.081716	0.096786	0.327104	-0.006386	-0.037362
Series11	1.422338	-0.634284	0.342219	0.989567	0.189125	0.415679	0.366245	-0.047839	-0.060504	0.191252	0.100336	0.053632
Series12	2.098947	0.916201	0.866814	0.248512	0.26752	-0.135769	-0.403409	-0.099703	-0.061635	-0.165724	-0.110404	-0.092618
Series13	2.024787	0.042945	0.709359	0.44374	0.33092	0.067731	-0.556837	-0.340761	0.062014	-0.322166	-0.12249	0.076262
Series14	1.954335	-0.029474	0.50485	1.151993	0.447127	-0.271188	-0.048397	0.178772	0.114923	-0.158597	-0.063588	0.049275
Series15	1.731455	0.389292	0.805259	0.931407	0.287265	-0.481005	0.032323	0.035392	-0.076855	-0.033531	-0.066702	0.035671

- LPC(Linear Predictive Coding) :
    - Linear Predictive Coding (LPC) is a technique used in digital signal processing and speech coding to represent the spectral envelope of a signal or speech.
    - The main goal of LPC is to model the vocal tract characteristics of speech signals. It aims to capture the formants, which are the resonant frequencies that contribute to the quality and timbre of speech sounds. By estimating and encoding the formants, LPC allows for efficient compression of speech signals while maintaining intelligibility.

- Application of LPC :
    - Speech and Audio Coding: LPC is extensively used in speech and audio compression algorithms such as speech codecs (e.g., GSM, AMR, G.729) and audio codecs (e.g., MP3, AAC). By using LPC, redundant information in speech or audio signals can be efficiently removed, resulting in lower bit rates and reduced storage requirements while maintaining acceptable audio quality.

    - Speech Synthesis: LPC is employed in text-to-speech synthesis systems to generate natural-sounding speech. By modeling the speech signal with an LPC analysis, it is possible to estimate the parameters of the vocal tract and use them to synthesize speech. LPC-based speech synthesis is commonly used in applications like voice assistants, automated announcements, and assistive technologies for individuals with speech impairments.

    - Speech Enhancement: In noisy environments, LPC can be employed for speech enhancement tasks such as noise reduction and dereverberation. By modeling the speech signal and the background noise, LPC-based algorithms can attenuate the noise components and improve the intelligibility of the speech.

    - Voice over IP (VoIP): LPC is utilized in VoIP applications to compress and transmit speech signals over network connections with limited bandwidth. By using LPC-based codecs, efficient voice communication can be achieved while minimizing the required network resources.