Project Report

Comparative Analysis of DNA sequences using Fourier Power Spectrum

**1.Introduction**

In the last few decades, several methods to classify genes and proteins have been proposed. Most of these methods are alignment-based in which optimal alignments are obtained by using selected scoring systems. These methods provide accurate classification of biological sequences, and several algorithms have been developed and successfully applied. Nevertheless, their major drawback is due to significantly high time and memory consumption which is not suitable when a quick clustering needs to be made. An alignment-free technique is a trending method that often gives much faster classification on the same dataset. For example, the k-mer method is among the most popular alignment-free methods. In order to measure how different, the two sequences are, the set of k-mers, or sub sequences of length k, in the two biological sequences are collected and then the evolutionary distance between them is computed. The k-mer method gives comparable results to alignment-based methods while being computationally faster.

Discrete Fourier Transform (DFT) is a powerful tool in signal and image processing. During recent years, DFT has been increasingly used in DNA research, such as gene prediction, protein coding region, genomic signature, hierarchical clustering, periodicity analysis.

A DFT power spectrum of a DNA sequence reflects the nucleotide distribution and periodic patterns of that sequence, and it has been applied to identify protein coding regions in genomic sequences. In this paper we provide a new alignment-free method to classify DNA sequences based on the DFT power spectrum. The method is tested and compared to other state-of-the-art methods on various datasets for speed and accuracy.

2.**Database**

Here, is the link which was used in that project –

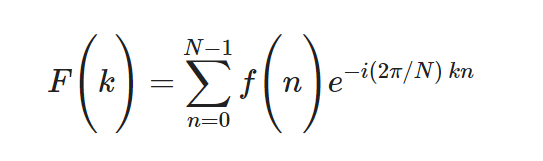
[**database URL**](https://drive.google.com/drive/folders/18pmx6JZ2WTquihNNZ1PkK3ecvhWh_pe0?usp=share_link)

**3.Methodology/Algorithm**

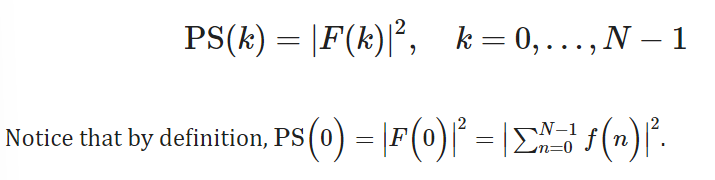
3.1 Mathematical Background

In signal processing, sequences in time domain are commonly transformed into frequency domain to make some important features visible. Via that transformation, no information is lost but some hidden properties could be revealed.

One of the most common transformations is Discrete Fourier Transform. For a signal of length N,f(n),n=0,…,N−1, the DFT of the signal at frequency k is

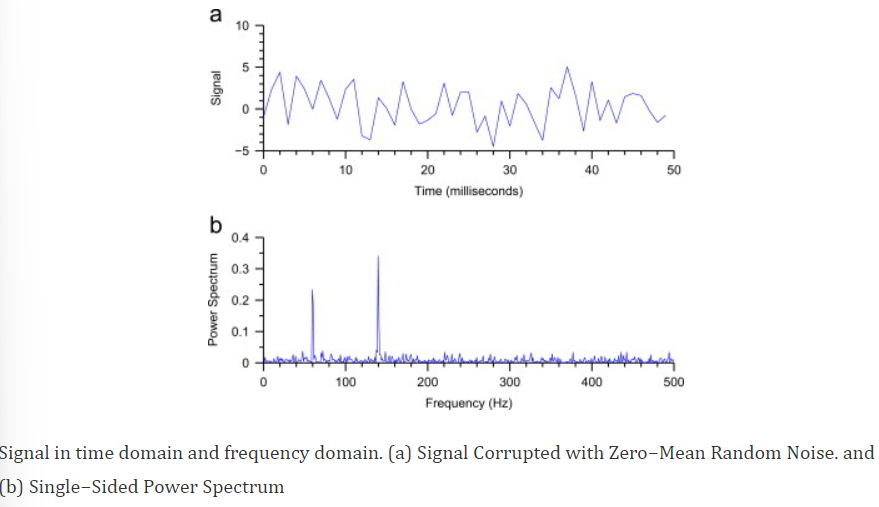


for k=0,…,N−1. The DFT power spectrum of a signal at frequency k is defined as



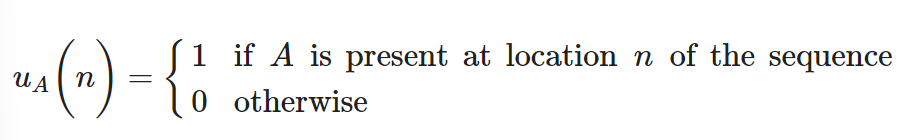
The DFT is often used to find the frequency components of a signal buried in a noisy time domain. For example, let *y* be a signal containing a 60 Hz sinusoid of amplitude 0.8 and a 140 Hz sinusoid of amplitude 1. This signal can be corrupted by a zero-mean random noise:

y=0.8⁎sin(2⁎π⁎60⁎t)+sin(2⁎π⁎140⁎t)+random



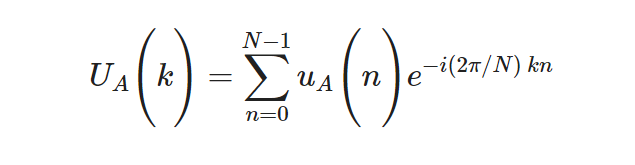
3.2. Moment vectors

For a DNA sequence composed of nucleotides adenine (A), cytosine (C), guanine (G), and thymine (T), one typical way to get numerical representation is to use binary indicator sequences. The values of these sequences are either 0 or 1 indicating the absence or presence of a specific nucleotide. Specifically, for a given DNA sequence of length *N*, we define *u* *A* of the same length as follows:

uC,uG,uT are defined similarly.

For example, for the sequence *AGTCTTACGA*, the corresponding indicator sequence of nucleotide *A* is *u* *A*=1000001001.

The DFT of *u* *A* is *U* *A* where

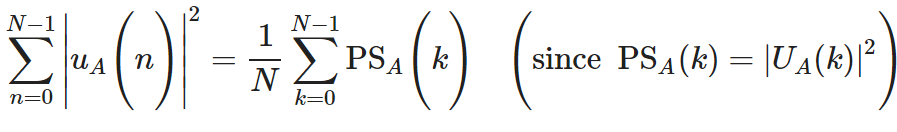


for k=0,…,N−1.

The DFT power spectrum of *u* *A* is PS*A* where PSA(k)=|UA(k)|2,k=0,…,N−1. The corresponding power spectrum for nucleotides C,G,T is defined similarly. In general, for a gene sequence of length *N*, let NA,NC,NG,NT be the number of nucleotide A,C,G,T in that sequence, respectively.

It is difficult to compare numerical sequences with different lengths, so we cannot cluster genes and genomes based on their power spectra sequences. One common approach to get over this problem is to use mathematical moments, e.g. for nucleotide *A* defines *j*th moment MAj=αAj∑N−1k=0(PSA(k))j,j=1,2… , where *α* *j* *A* be scaling factors. We want higher moments to converge to zero, i.e. essential information is kept in the first few moments. Thus, the chosen normalization factors *α* *j* *A* must reflect the nature of the sequences. Let us examine the binary indicator sequence of one nucleotide, *A*, in more detail.

By Parseval׳s theorem,

The left side is actually *N* *A*, i.e. the number of 1 in the *A* binary sequence. Hence, ∑N−1k=0PSA(k)=NAN. So it is reasonable for *α* *j* *A* to be a power of *N* *AN*. As stated above, we want moments converge to zero gradually so that information loss is minimal, thus αAj=1/(NAN)j−1 is the best choice (which will be verified later). Therefore

MAj=1Nj−1ANj−1∑k=0N−1(PSA(k))j

With this normalization, MA1=∑N−1k=0PSA(k)=NAN. Our experimental results on various datasets proved that this is a good normalization. However, by re-examining the formula, we find that a slight modification can be made to get better outcomes. we know PSA(0)=∣∣FA(0)∣∣2=∣∣∑N−1n=0uA(n)∣∣2=N2A. Thus PSA(0) might hold large weight compared to PS*A*(*k*) for other index *k*, which in turn leads to unnecessary memory consumption and computations for higher moments. Therefore, the terms PSA(0) are removed from the moments, and *M* 1 *A* becomes ∑N−1k=1PSA(k)=NAN−PSA(0)=NAN−N2A=NA(N−NA). From the first modified moment, we know how to adjust the scaling factor for the j-th moment in general, i.e. *α* *j* *A* must be a power of NA(N−NA). Thus the new normalization is

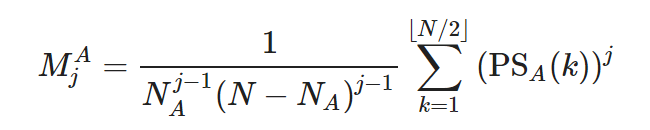
MAj=1Nj−1A(N−NA)j−1∑k=1N−1(PSA(k))j

The fact that higher moments tend to zero is verified as follows:

MAj=NA(N−NA)∑k=1N−1(PSA(k)NA(N−NA))j=NA(N−NA)∑k=1N−1zjk

where zk=PSA(k)/NA(N−NA). Notice that ∑N−1k=1zk=1, thus it is obvious that limj→∞∑N−1k=1zjk=0.

This fact also shows that αAj=1/(NA(N−NA))j−1 is the best scaling factor, as for αAj=1/(NA(N−NA))j, MA1=1 so moments tend to zero very fast, thus much information can be lost, and for αAj=1/(NA(N−NA))j−2, MA1=N2A(N−NA)2 so moments tend to zero much slower, thus more computational time and memory storage are needed. Additionally, due to symmetric property of DFT coefficients, we only have to consider the first half of power spectrum. Therefore, the moments are improved as follows:



The moments for other nucleotides C,G,T are given similarly. Then the first few moments are used to construct vectors in Euclidean space. Our experimental results show that three moments are sufficient for an accurate clustering. Thus, each gene or genome sequence can be realized as a geometric point in the 12-dimensional Euclidean space, i.e. (MA1,MC1,MG1,MT1,MA2,MC2,MG2,MT2,MA3,MC3,MG3,MT3). Pairwise Euclidean distances between these points are calculated to cluster the gene or genome sequences. We call this Power Spectrum Moments method, or PS-M method.

Matlab code :

##Main.m code

clear all ;

close all ;

set(0,'DefaultFigureWindowStyle','normal');

drawnow;

clc;

condition = true;

while condition

choice = menu('Please choose a data set!',...

' 1. Mammals', ...

' 2. Influenza A virus', ...

' 3. Coronavirus', ...

' 4. HRV', ...

' 5. Bacteria', ...

' 6. Quit');

switch choice

case 1

TestUPGMA('Mammals');

case 2

TestUPGMA('Influenza');

case 3

TestUPGMA('Corona');

case 4

TestUPGMA('HRV');

case 5

TestUPGMA('Bacteria');

case 6

msgbox('You closed the menu!');

condition = false;

end

end

fprintf('\n');

## GetMomentVectorPs.m code:

function [v] = GetMomentVectorPS(seq)

%UNTITLED2 Summary of this function goes here

% Detailed explanation goes here

n=length(seq);

%Binary indicator sequences of A, C, G, T (each value will be either 1 or 0

%; 1 if that nucleotide appears in that position and 0 otherwise)

uA=zeros(1, n);

uC=zeros(1, n);

uG=zeros(1, n);

uT=zeros(1, n);

%Sequences' length (of A, C, G, T respectively)

nA=0;

nC=0;

nG=0;

nT=0;

%Get the binary indicator sequences and their lengths

for i=1:n

nu=seq(i);

switch upper(nu)

case {'A'}

uA(i)=1;

uC(i)=0;

uG(i)=0;

uT(i)=0;

nA=nA+1;

case {'C'}

uC(i)=1;

uA(i)=0;

uG(i)=0;

uT(i)=0;

nC=nC+1;

case {'G'}

uG(i)=1;

uA(i)=0;

uC(i)=0;

uT(i)=0;

nG=nG+1;

case {'T'}

uT(i)=1;

uA(i)=0;

uC(i)=0;

uG(i)=0;

nT=nT+1;

end

end

%Discrete Fourier Transforms

UA=fft(uA);

UC=fft(uC);

UG=fft(uG);

UT=fft(uT);

%Exclude the first term

UA(1)=[];

UC(1)=[];

UG(1)=[];

UT(1)=[];

% Get the first half of the transform (since it's symmetric)

m=floor((n-1)/2);

UA1=UA(1:m);

UC1=UC(1:m);

UG1=UG(1:m);

UT1=UT(1:m);

%Power spectrums

PSA=abs(UA).^2;

PSC=abs(UC).^2;

PSG=abs(UG).^2;

PST=abs(UT).^2;

%Normalized moments

MA=zeros(1,3);

MC=zeros(1,3);

MG=zeros(1,3);

MT=zeros(1,3);

%Moment vector

for j=1:3

MA(j)=(nA\*(n-nA))sum(PSA.^j)/(nA^(j)(n-nA)^(j));

MC(j)=(nC\*(n-nC))sum(PSC.^j)/(nC^(j)(n-nC)^(j));

MG(j)=(nG\*(n-nG))sum(PSG.^j)/(nG^(j)(n-nG)^(j));

MT(j)=(nT\*(n-nT))sum(PST.^j)/(nT^(j)(n-nT)^(j));

end

v=[MA(1), MC(1), MG(1), MT(1), MA(2), MC(2), MG(2), MT(2), MA(3), MC(3), MG(3), MT(3)];

end

### getBacteriaData.m code:

clear

data={

'AE005674.1' 'Shi-fle-2a'

'CP001383.1' 'Shi-fle-17'

'CP001671.1' 'Ecoil-ABU'

'CP000468.1' 'Ecoil-APEC'

'AE009952.1' 'Yer-KIM'

'AL590842.1' 'Yer-CO92'

'CP001593.1' 'Yer-Z176003'

'CP001585.1' 'Yer-D106004'

'AM295250.1' 'Sta-car'

'AE015929.1' 'Sta-epi-AT'

'AP006716.1' 'Sta-hae'

'CP001837.1' 'Sta-lug'

'CP001151.1' 'Rho-KD131'

'CP000578.1' 'Rho-ATCC'

'AM260480.1' 'Ral-H16'

'CP000091.1' 'Ral-JMP'

'CP001598.1' 'Bac-A0248'

'AE016879.1' 'Bac-Ames'

'CP001215.1' 'Bac-CDC'

'AE017225.1' 'Bac-Stern'

'CP000048.1' 'Bor-hermsii'

'CP000976.1' 'Bor-dut'

'CP000049.1' 'Bor-tur'

'CP000993.1' 'Bor-recu'

'CP000246.1' 'Clo-per-ATCC'

'CP000312.1' 'Clo-per-SM101'

'BA000016.3' 'Clo-per-13'

'CP000527.1' 'Des-vul-DP4'

'CP002297.1' 'Des-vul-RCH1'

'AE017285.1' 'Des-vul-Hild'

};

len=length(data);

for i=1:len

seqs(i).Header=data{i,2};

seqs(i).Sequence = getgenbank(data{i,1}, 'SequenceOnly', true);

end

fastawrite('Bacteria.fasta', seqs);

### getEdistance.m code:

function [EDist] = getEDistance(A,B)

% Get euclidean distance

% Number of column of A and B are dimensional, must be same

% Rows is the data points

%8/2/2013 C. Yin.

%A=[1 2 3 -1;4 5 6 -1] % The 4-D space, two points in A

%B=[-1 3 2 4;-2 5 4 -1] % The 4-D space, two points in B

%A=[1 2 3;4 5 3] % The 3-D space, two points in A

%B=[-1 3 2;3 4 -1] % The 3-D space, two points in B

diffSquare=(A-B).^2;

EDist=sqrt(sum(diffSquare,2));

end

#### TestUPGMA.m code:

function TestUPGMA(name)

%UNTITLED Summary of this function goes here

% Detailed explanation goes here

tStart = tic;

file=strcat(name,'.fasta');

seqs = fastaread(file);

len = length(seqs);

for i = 1:len

lenX(i)=length(seqs(i).Sequence);

end

lenX

b=max(lenX);

fprintf('Min: %d \n', min(lenX));

fprintf('Max: %d \n', max(lenX));

%Get moment vectors

for i=1:len

v{i}=1/b\*GetMomentVectorPS(seqs(i).Sequence);

end

%Get (Euclidean) lower triangular distance matrix based on above moment vectors

for j=1:len

for i=j:len

D(i,j)=getEDistance(v{i}, v{j});

end

end

%Rearrange the above distance matrix into a row vector in order to

%use seqlinkage

d=squareform(D);

%Phylogenetic tree

tree= seqlinkage(d,'average',seqs);

h = plot(tree, 'orient', 'left');

title('Similarity distance using our new method with UPGMA', 'FontName', 'AvantGarde','FontSize', 10,'FontWeight','bold')

tEnd = toc(tStart);

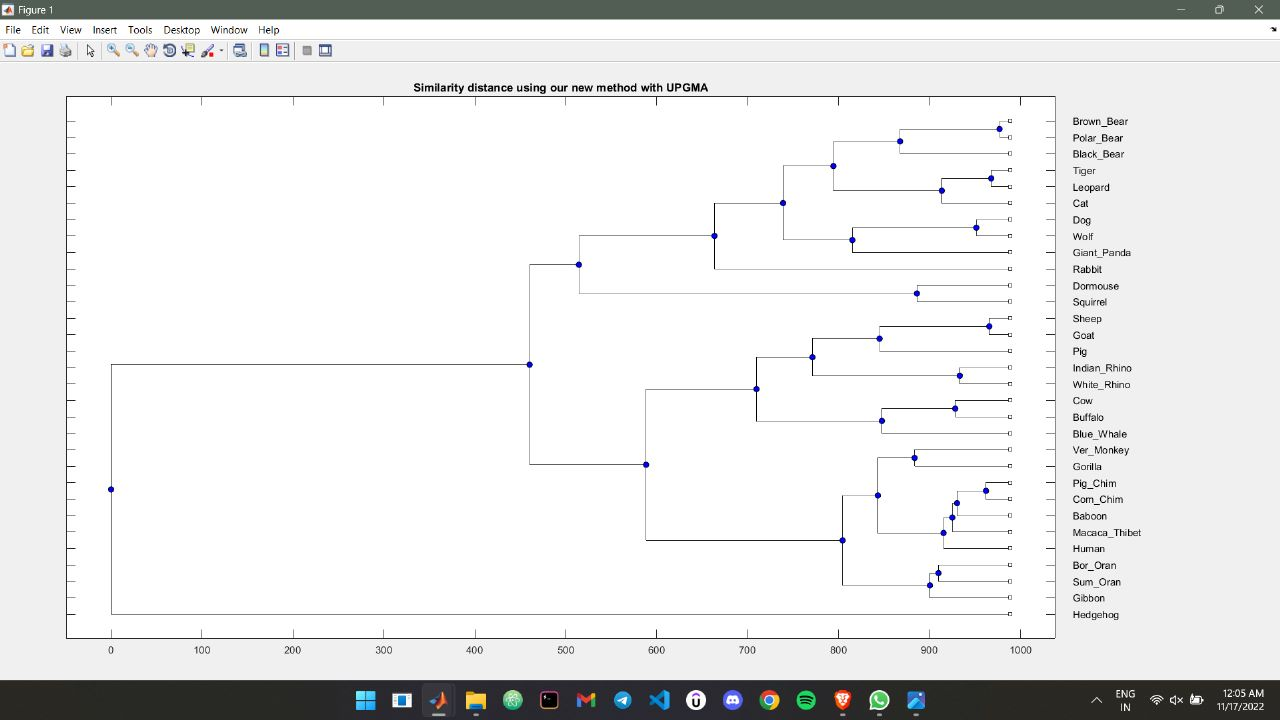
fprintf('%d minutes and %f seconds\n',floor(tEnd/60),rem(tEnd,60));

end

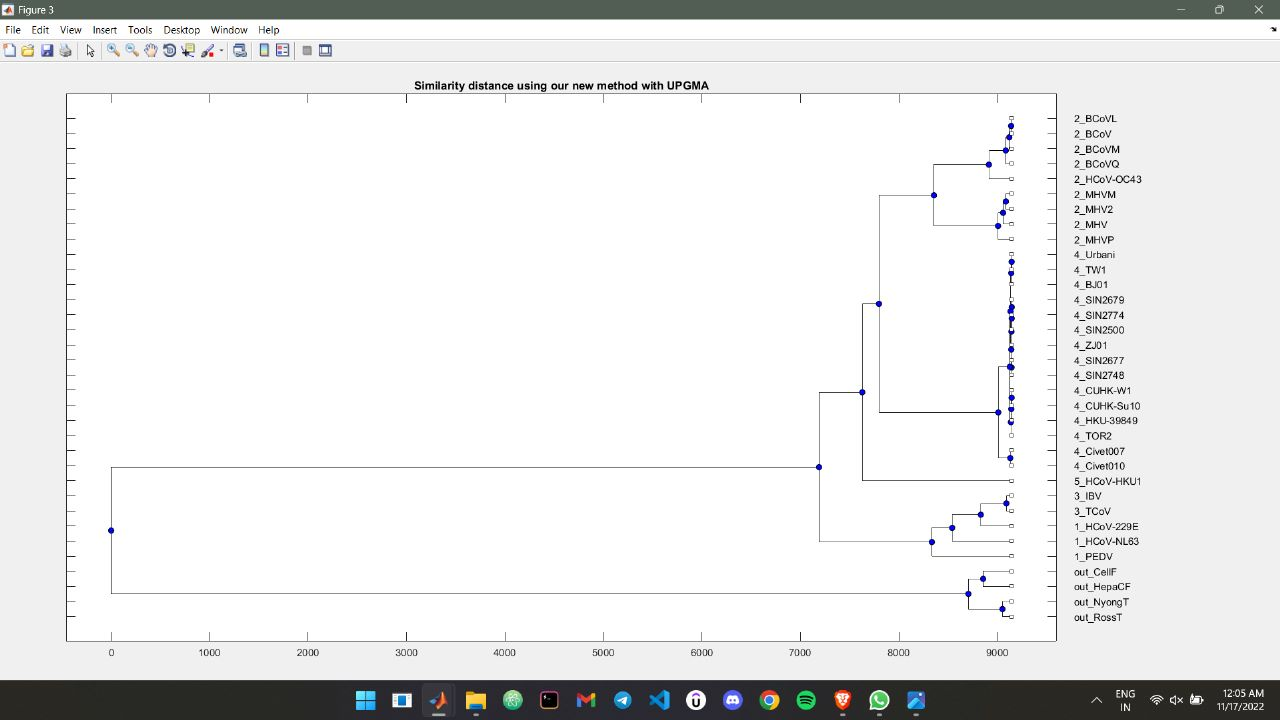
**Result :**

The PS-M method is tested on different datasets that range from small to medium size, as well as short to long genomes. In order to compare and analyze various genomic data, moment vectors are calculated and a matrix of Euclidean pairwise distances between those vectors is constructed. To cluster data into biological groups, a phylogenetic tree is built based on the distance matrix using the UPGMA method

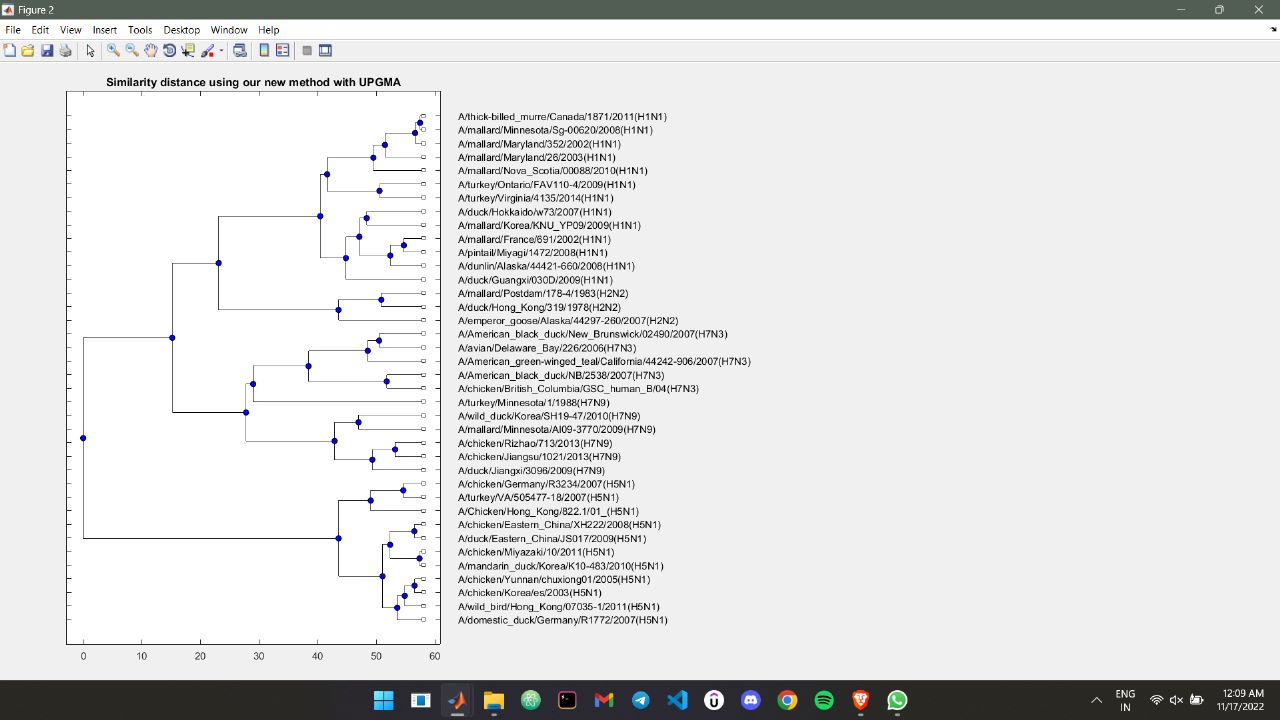
Mammals –



CoronaVirus –



Influenza A Virus –



**Conclusion :**

In this work, we have established an effective similarity measure of DNA

sequences based on DFT. We first performed DFT on DNA sequence after

converting symbolic sequences to four binary indicator sequences. Euclidean

distance is used to calculate the similarity of DFT coefficients. We conducted different DNA sequence mutants and assess the accuracy of the new

DFT metric on the mutants. The similarity metrics have been evaluated by

constructing phylogenetic trees of virus at gene levels. Our work provides

encouraging steps for applying the rediscovered DFT approach for effective

comparative studies on biological sequences.

**References:**

**Hoang T, Yin C, Zheng H, Yu C, Lucy He R, Yau SS. A new method to cluster DNA sequences using Fourier power spectrum. J Theor Biol. 2015 May 7;372:135-45. doi: 10.1016/j.jtbi.2015.02.026. Epub 2015 Mar 5. PMID: 25747773; PMCID: PMC7094126.**