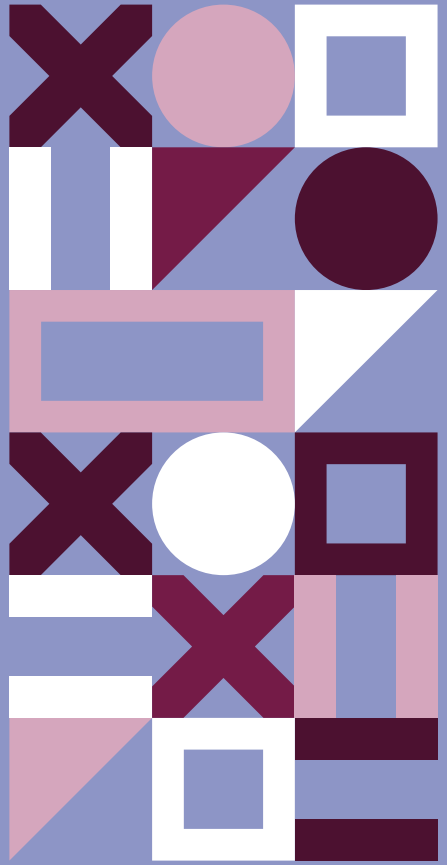


CHAOS GAME REPRESENTATION SIMILARITY ANALYSIS

TEAM 15

Deepthi Sudharsan

CB.EN.U4AIE19022

Isha Indhu S

CB.EN.U4AIE19030

Lakshaya Karthikeyan

CB.EN.U4AIE19039

Roshan Tushar S

CB.EN.U4AIE19071

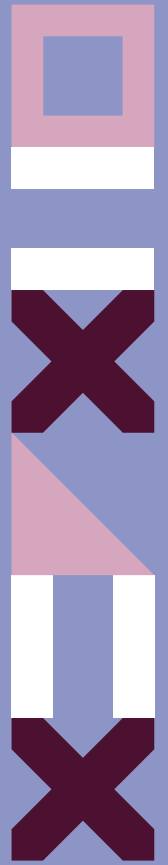


TABLE OF CONTENTS:

Theory of CGR

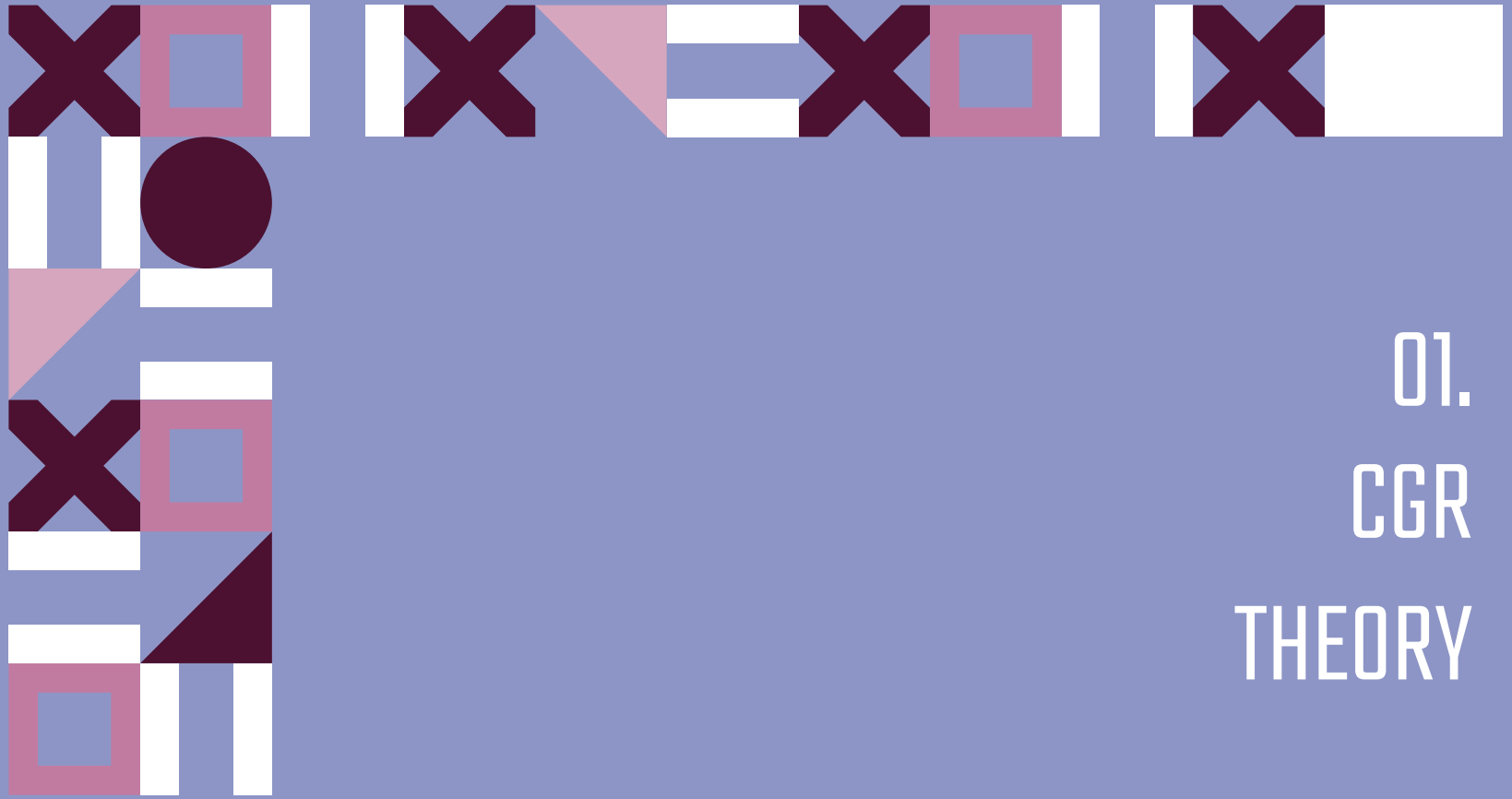
Explaining the workings of Chaos Game Representation

FCGR

Analysis of sequences using Frequency - CGR

CCGR

Analysis of sequences using Coordinate - CGR

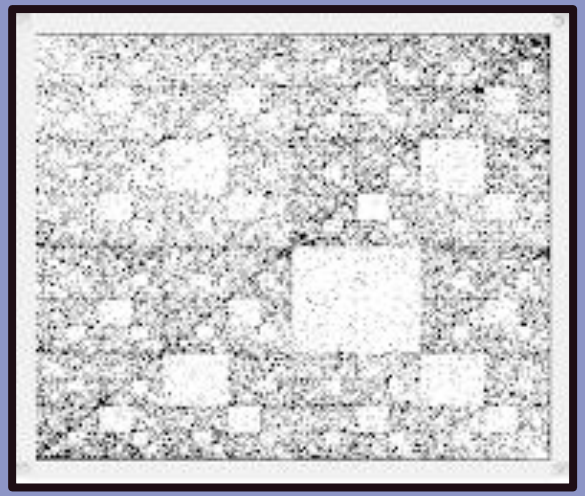


01.
CGR
THEORY

THEORY OF CGR:

- Graphical representation of a sequence
- Long 1D sequence → Graphical Form
- Aids us in recognizing patterns

- Mainly used for genome sequence encoding and classification
- Can be used to identify genome fragments



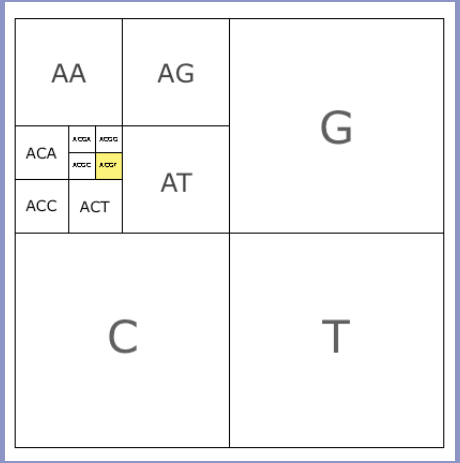
02.

FREQUENCY

CGR

FCGR FOR GENOMES:

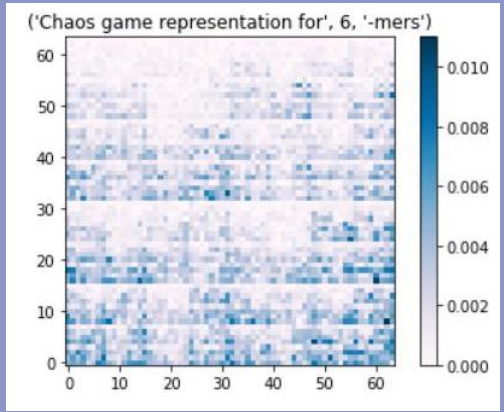
- In case of genome data, the CGR image contains 4^k boxes (k = length of kmers)
- The boxes are colored:
  - ❑ from light (kmer: not present)
  - ❑ to dark (kmer: most abundant)



For each nucleotide in a kmer, the image is subdivided into 4 quadrants:

- ➢ A in the top left
- ➢ T in the top right
- ➢ C in the bottom left
- ➢ G in the bottom right

Each quadrant is split according to the same principle for the next nucleotide in the kmer, *recursively*



CODE

def count_kmer():

```
def count_kmers(self,sequence, k):
    d = collections.defaultdict(int)
    for i in range(len(sequence)-(k-1)):
        d[sequence[i:i+k]] +=1
    for key in list(d):
        if "N" in key:
            del d[key]
    return d
```

- Takes the sequence & k as inputs
- Default dictionary creation for count
- Splitting the sequence into kmers
- Storing the kmers along with the number of occurrences
- Removing key in case “N” is present (N for unknown nucleic acid residue)

CODE

def probabilities():

- Default dictionary creation to store probabilities
- Calculates the probabilities of each kmer using the count of kmers

```
def probabilities(self,kmer_count, k):
    probabilities = collections.defaultdict(float)
    N = len(kmer_count)
    for key, value in kmer_count.items():
        # calculating the probablities for each k-mer in dictionary
        probabilities[key] = float(value) / (N)
    return probabilities
```

CODE

def chaos_game_representation():

```python
def chaos_game_representation(self,probabilities, k):
    # FCGR matrix size
    array_size = int(math.sqrt(4**k))
    chaos = []
    for i in range(array_size):
        # creating a 2D square matrix of size array_size with zeros
        chaos.append([0]*array_size)

    maxx = array_size
    maxy = array_size
    posx = 0
    posy = 0
    # finding the grids to which the k-mers belong
    for key, value in probabilities.items():
        for char in key:
            if char == "T":
                posx += maxx // 2
            elif char == "C":
                posy += maxy // 2
            elif char == "G":
                posx += maxx // 2
                posy += maxy // 2
            maxx = maxx // 2
            maxy = maxy//2
        chaos[posy-4][posx-4] = value
        maxx = array_size
        maxy = array_size
        posx = 4
        posy = 4

    return chaos
```

- Creating a 2D array of size $(\sqrt{4^k}, \sqrt{4^k})$
- Initialising the chaos matrix with zeros
- Finding the coordinates of the grids to which the kmers belong
- Storing the probabilities of the kmers in their corresponding grids

CODE

def CGR_prob_dist():

```
# function to find the Euclidean distance between two probablity matrices
def CGR_prob_dist(chaos1,chaos2):
    dist = np.square(np.sum(np.square(np.array(chaos1) - np.array(chaos2))))
    return dist
```

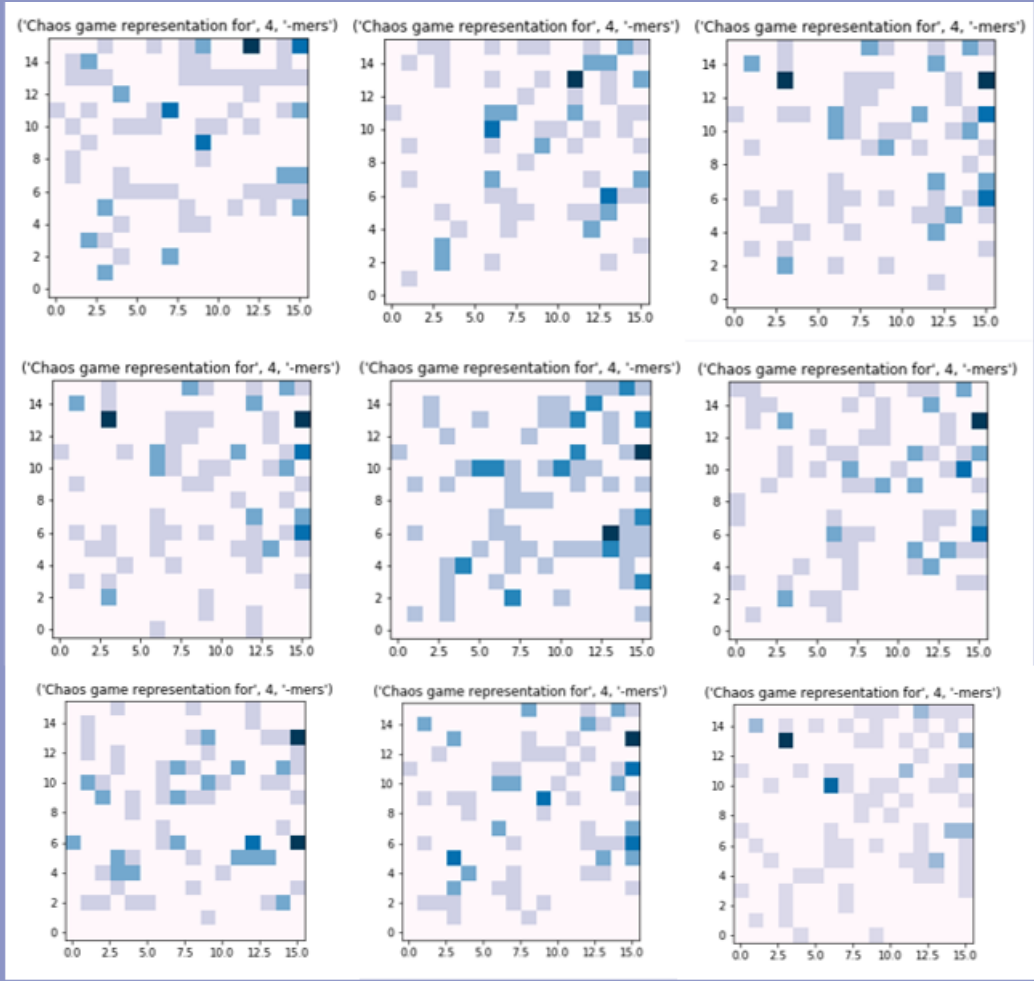
- Calculating Euclidean distance between 2 chaos probability matrices

$$\sqrt{\sum_{i=0}^{n}\sum_{j=0}^{n}\left(p_{1ij}-p_{2ij}\right)^2}$$

READING THE FASTA FILES & PLOTTING THE CGR MATRICES:

```
import glob
# reading all fasta files in folder
f = glob.glob("C:\\Users\\Lakshaya Karthikeyan\\Documents\\University\\SEM 4\\IBS - 4\\Project\\ANIMAL_GENOME\\*.fasta")
```

```
k = 4                                   # kmer length
chaos_mat = []
for i in range(0,len(f)):               # for loop to find and plot the CGR mat for all the files in the folder
    f1 = f[i]
    print(f1[f1.rindex('\\')+1:len(f1)])
    chaos_mat.append(CGR_plot(f[i],k))
```



FCGR PLOTS:

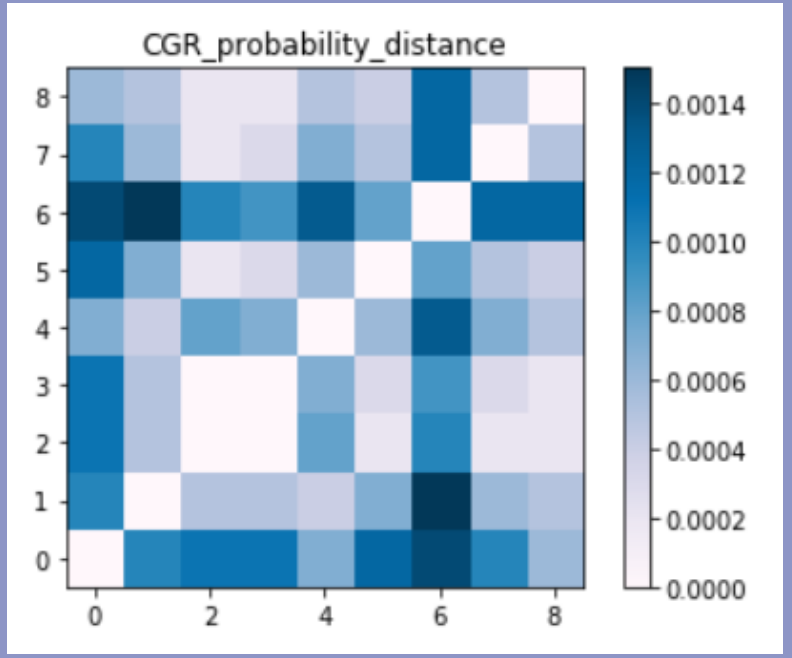
Gallus
Human
Opossum

Goat
Lemur
Rabbit

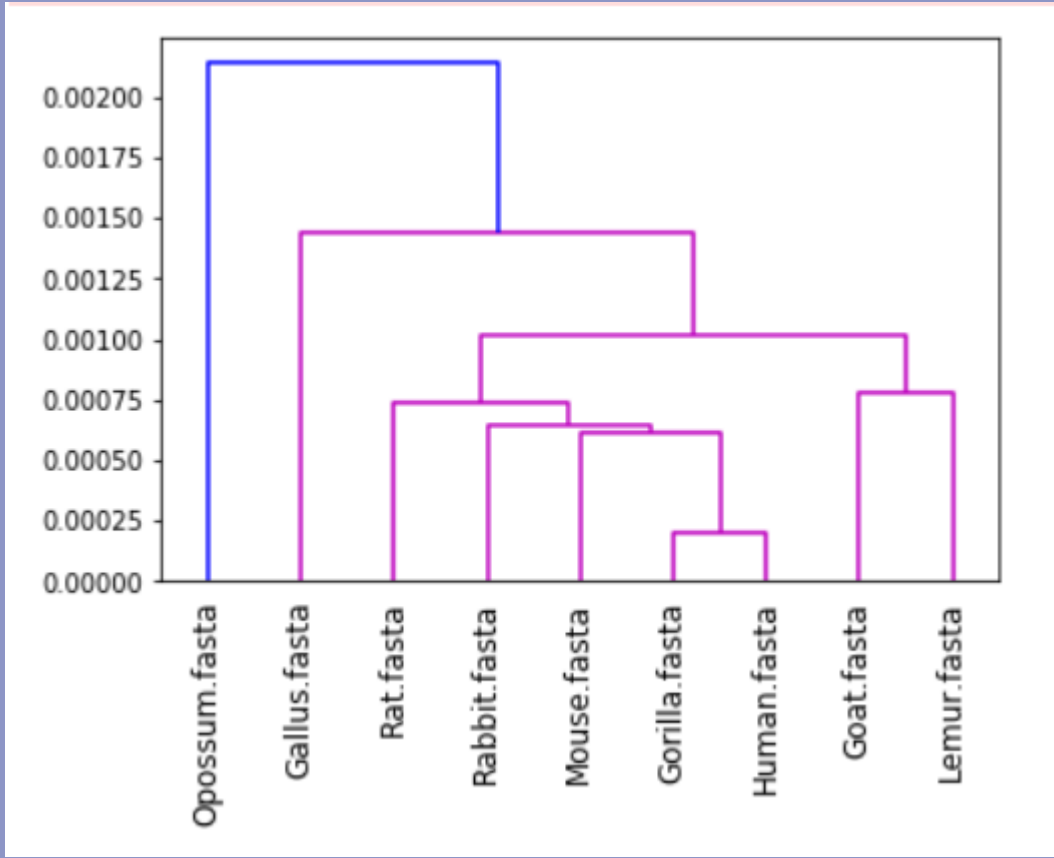
Gorilla
Mouse
Rat

FCGR DISTANCE PLOTS:

	0	1	2	3	4	5	6	7	8
0	0.0000	0.0010	0.0011	0.0011	0.0007	0.0012	0.0014	0.0010	0.0006
1	0.0010	0.0000	0.0005	0.0005	0.0004	0.0007	0.0015	0.0006	0.0005
2	0.0011	0.0005	0.0000	0.0000	0.0008	0.0002	0.0010	0.0002	0.0002
3	0.0011	0.0005	0.0000	0.0000	0.0007	0.0003	0.0009	0.0003	0.0002
4	0.0007	0.0004	0.0008	0.0007	0.0000	0.0006	0.0013	0.0007	0.0005
5	0.0012	0.0007	0.0002	0.0003	0.0006	0.0000	0.0008	0.0005	0.0004
6	0.0014	0.0015	0.0010	0.0009	0.0013	0.0008	0.0000	0.0012	0.0012
7	0.0010	0.0006	0.0002	0.0003	0.0007	0.0005	0.0012	0.0000	0.0005
8	0.0006	0.0005	0.0002	0.0002	0.0005	0.0004	0.0012	0.0005	0.0000

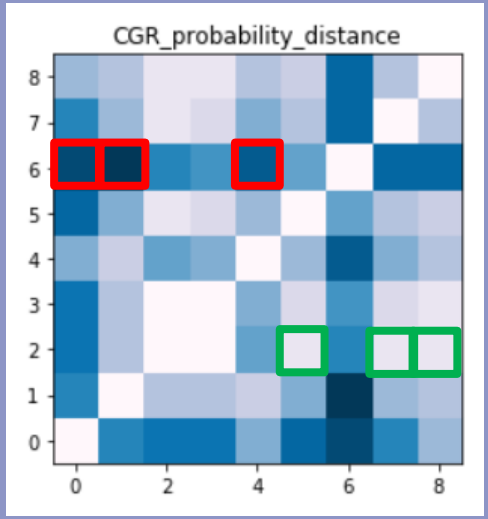


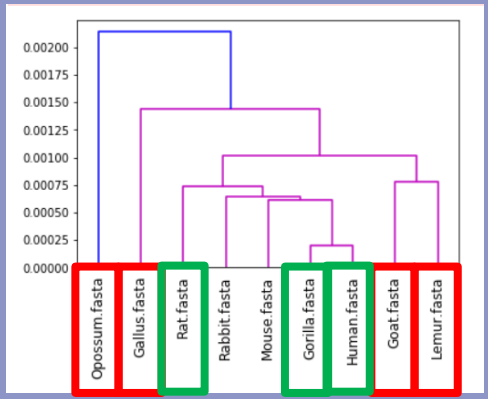
DENDROGRAM:



The relationship between the various animal sequences is evident in this hierarchical representation

We can observe that humans and gorillas are the most closely related species





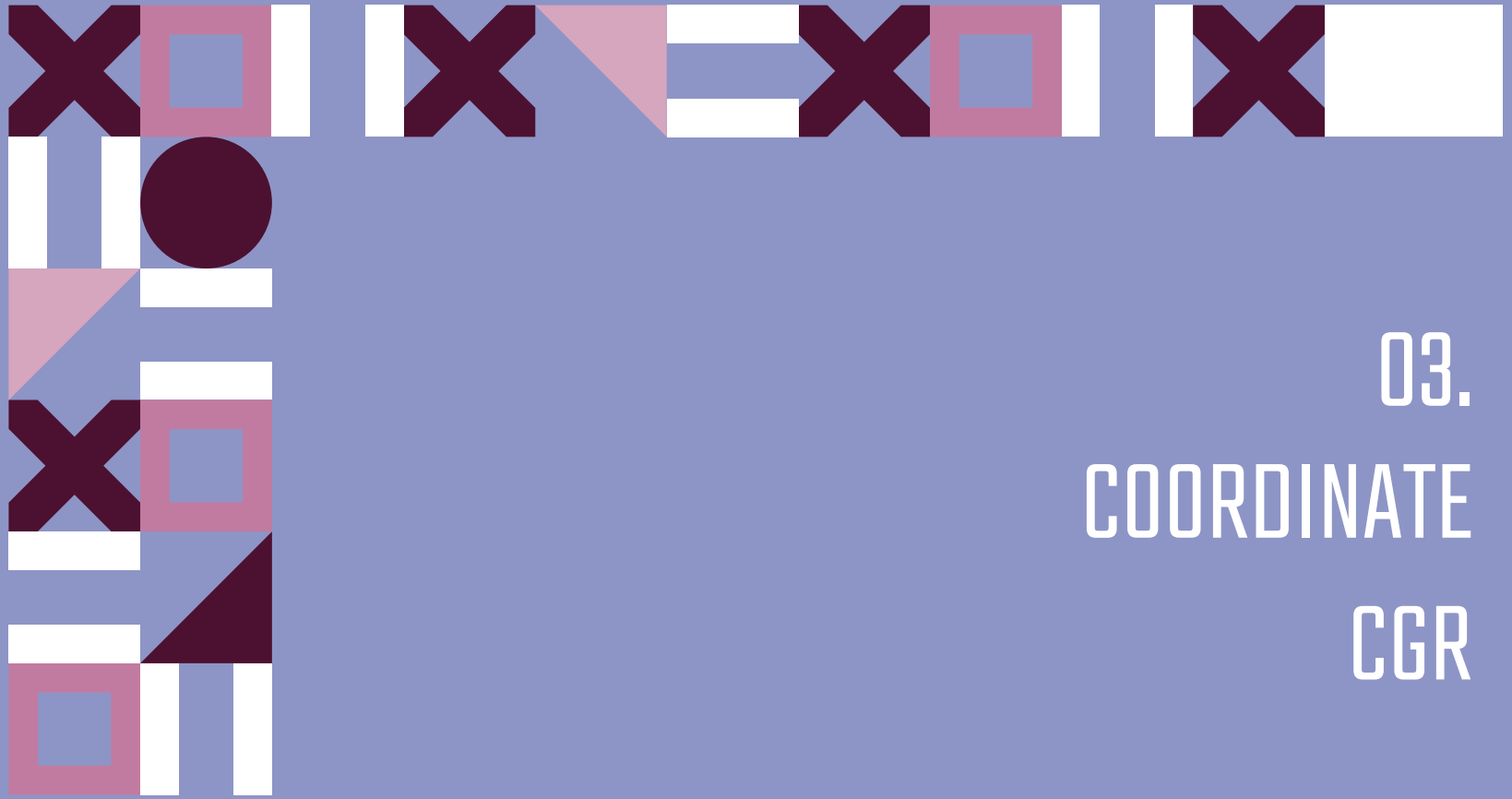
OBSERVATIONS:

LEAST CLOSELY RELATED:

Animal 1	Animal 2	Distance
Opossum	Goat	0.0015
Gallus	Opossum	0.0014
Opossum	Lemur	0.0013

MOST CLOSELY RELATED:

Animal 1	Animal 2	Distance
Gorilla	Human	0.0000
Gorilla	Rabbit	0.0002
Rat	Gorilla	0.0002

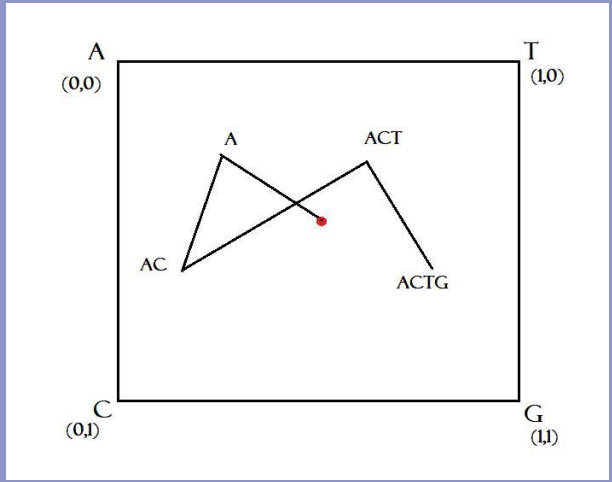


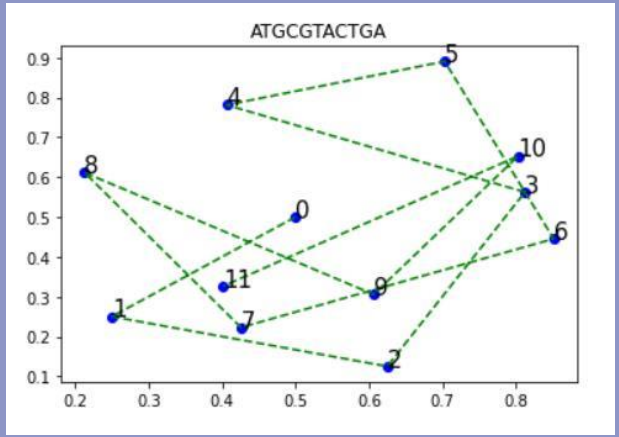
03.
COORDINATE
CGR

CCGR FOR GENOMES:

- Steps followed for coordinate CGR:
  1) Start from the center of the grid
  2) 1st coordinate → plotted halfway between the center of the square and the vertex representing this nucleotide (A)
  3) Successive coordinates → plotted halfway between the previous point and the vertex representing the current nucleotide

$$X_i = 0.5(X_{i-1} + g_{ix})$$
$$Y_i = 0.5(Y_{i-1} + g_{iv})$$





CODE

def CGR_coord():

```
x = {'A': 0,'C': 0,'G':1,'T':1}
y = { 'A': 0,'C': 1,'G':1,'T':0}

A = (0,0)
C = (0,1)
G = (1,1)
T = (1,0)

#Function to plot the CGR co-ordinate methods
def CGR_coord(s1):
    CGR_x = [0.5]
    CGR_y = [0.5]
    for s in s1:
        s = s.upper()
        tempx = CGR_x[-1]
        tempy = CGR_y[-1]
        CGR_x.append(tempx - 0.5*(tempx - x[s]))
        CGR_y.append(tempy - 0.5*(tempy - y[s]))
    return CGR_x,CGR_y
```

- Initialising the coordinates of the nucleotides
- Coordinates are computed using the following formula:

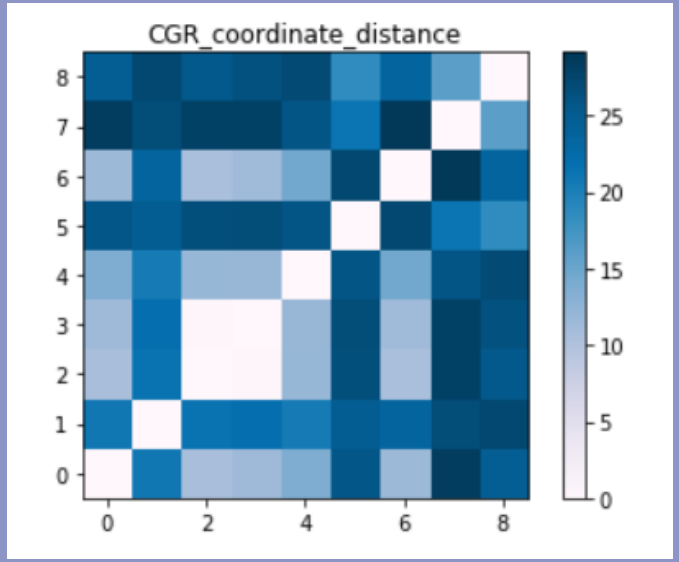
$$X_i = 0.5(X_{i-1} + g_{ix})$$
$$Y_i = 0.5(Y_{i-1} + g_{iv})$$

CODE

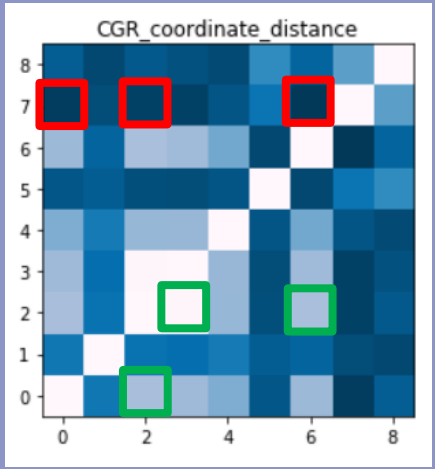
def probabilities():

```python
# Function to find the euclidean distance between the two CGR coordinates
def Dist_CGR_coord(CGR_x1,CGR_x2,CGR_y1,CGR_y2):
    l = min(len(CGR_x1),len(CGR_x2))
    X_diff = abs(np.array(CGR_x1[0:l]) - np.array(CGR_x2[0:l]))
    Y_diff = abs(np.array(CGR_y1[0:l]) - np.array(CGR_y2[0:l]))
    return np.sqrt(sum(np.square(X_diff)) + sum(np.square(Y_diff)))
```

```python
dist_cgr_coord = np.zeros((len(f),len(f)))
for i in range(0,len(f)):
    for j in range(0,len(f)):
        if(i!=j):
            CGR_x1,CGR_y1 = CGR_coord(readf(f[i]))
            CGR_x2,CGR_y2 = CGR_coord(readf(f[j]))
            dist_cgr_coord[i][j] = Dist_CGR_coord(CGR_x1,CGR_x2,CGR_y1,CGR_y2)
```



OBSERVATIONS:



	0	1	2	3	4	5	6	7	8
0	0.000000	4.590395	3.267654	3.386282	3.713644	5.073672	3.397166	5.346775	4.966942
1	4.590395	0.000000	4.628751	4.699545	4.537484	4.993449	4.856841	5.183881	5.239043
2	3.267654	4.628751	0.000000	0.577350	3.464200	5.154723	3.250600	5.303029	5.061594
3	3.386282	4.699545	0.577350	0.000000	3.457662	5.174762	3.364177	5.296089	5.139173
4	3.713644	4.537484	3.464200	3.457662	0.000000	5.093036	3.832494	5.085949	5.214009
5	5.073672	4.993449	5.154723	5.174762	5.093036	0.000000	5.236972	4.617563	4.311184
6	3.397166	4.856841	3.250600	3.364177	3.832494	5.236972	0.000000	5.399882	4.861818
7	5.346775	5.183881	5.303029	5.296089	5.085949	4.617563	5.399882	0.000000	4.007331
8	4.966942	5.239043	5.061594	5.139173	5.214009	4.311184	4.861818	4.007331	0.000000

LEAST CLOSELY RELATED:

Animal 1	Animal 2	Distance
Opossum	Rabbit	5.399882
Gallus	Rabbit	5.346775
Gorilla	Rabbit	5.303029

MOST CLOSELY RELATED:

Animal 1	Animal 2	Distance
Gorilla	Human	0.577350
Gorilla	Opossum	3.250600
Gallus	Gorilla	3.267654

WEB GUI

Created By:

Deepthi Sudharsan

Isha Indhu S

Lakshaya Karthikeyan

Roshan Tushar S

IBS4 - Team 15

Chaos Game Representation

Finding Similarity Between Sequences

Select the method of input:

Radio

- (•) Paste the Sequences
- ( ) Upload the Sequences (.FASTA format)
- ( ) Select from existing files

Sequence 1

Sequence 2

THANK YOU

SCAN FOR GITHUB