**Text to Binary**

def BreakingMessage(binmess, k, messlen):

i = 0

messarr = []

while messlen >= k :

messarr.append(binmess[i:i+k])

i = i+k

messlen = messlen-k

if(messlen > 0):

messarr.append(binmess[i:])

return messarr

**Basic GUI**

window = tkinter.Tk()

window.title("LDPC Transmitter")

window.geometry("900x700")

label = tkinter.Label(window, text = "Transmitter", font = ('arial', 25, 'bold'), pady=10).pack()

messlab = tkinter.Label(window, text = "Enter the message to be transmitted:", font = ('arial', 20, 'bold'), pady=5).pack()

mess = tkinter.Entry(window, font = ('arial', 25, 'bold'), width = 20, bg = "#ccc", bd = 3)

mess.pack()

tkinter.Button(window, text = "SEND IT", command = calhmat, bd=10, font = ('ubuntu', 15, 'bold'), pady=5).pack()

window.mainloop()

**Encoding**

def calhmat():

binmess = ''.join(format(ord(x), 'b') for x in mess.get())

temp = "\nMessage '"+str(mess.get())+"' in Binary format is: "

tkinter.Label(window, text = temp, font = ('ubuntu', 15, 'bold')).pack()

tkinter.Label(window, text = binmess, font = ('ubuntu', 15, 'bold')).pack()

num = 15 # Number of columns

dv = 4 # Number of ones per column, must be lower than d\_c (because H must have more rows than columns)

dc = 5 # Number of ones per row, must divide n (because if H has m rows: m\*d\_c = n\*d\_v (compute number of ones in H))

# H Matrix

H = pyldpc.RegularH(num,dv,dc)

# G Matrix

tG = pyldpc.CodingMatrix(H)

n,k = tG.shape # n = 15, k = 6

snr = 8

messlen = len(binmess)

messarr = BreakingMessage(binmess, k, messlen)

print("Messarr = ", messarr)

op = CodingMessage(messarr, tG, snr, k)

print(op)

temp2 = "Modulates Message to be sent is :\n "+str(op)

tkinter.Label(window, text = temp2, font = ('ubuntu', 15, 'bold')).pack()

**Code for Parity Check Matrix:**

def RegularH(n,d\_v,d\_c):

‘’’

Builds a regular Parity-Check Matrix H (n,d\_v,d\_c) following Callager's algorithm :

Paramaeters:

n: Number of columns (Same as number of coding bits)

d\_v: number of ones per column (number of parity-check equations including a certain variable)

d\_c: number of ones per row (number of variables participating in a certain parity-check equation);

Errors:

The number of ones in the matrix is the same no matter how we calculate it (rows or columns), therefore, if m is

the number of rows in the matrix:

m\*d\_c = n\*d\_v with m < n (because H is a decoding matrix) => Parameters must verify:

0 - all integer parameters

1 - d\_v < d\_v

2 - d\_c divides n

Returns: 2D-array (shape = (m,n))

‘’’

if n%d\_c:

raise ValueError('d\_c must divide n. Help(RegularH) for more info.')

if d\_c <= d\_v:

raise ValueError('d\_c must be greater than d\_v. Help(RegularH) for more info.')

m = (n\*d\_v)// d\_c

Set=np.zeros((m//d\_v,n),dtype=int)

a=m//d\_v

# Filling the first set with consecutive ones in each row of the set

for i in range(a):

for j in range(i\*d\_c,(i+1)\*d\_c):

Set[i,j]=1

#Create list of Sets and append the first reference set

Sets=[]

Sets.append(Set.tolist())

#Create remaining sets by permutations of the first set's columns:

i=1

for i in range(1,d\_v):

newSet = np.transpose(np.random.permutation(np.transpose(Set))).tolist()

Sets.append(newSet)

#Returns concatenated list of sest:

H = np.concatenate(Sets)

return H

**Code for CodingMatrix:**

def CodingMatrix(MATRIX, use\_sparse=1):

"""

CAUTION: RETURNS tG TRANSPOSED CODING MATRIX.

Function Applies GaussJordan Algorithm on Columns and rows of MATRIX in order

to permute Basis Change matrix using Matrix Equivalence.

Let A be the treated Matrix. refAref the double row reduced echelon Matrix.

refAref has the form:

(e.g) : |1 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 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|

First, let P1 Q1 invertible matrices: P1.A.Q1 = refAref

We would like to calculate:

P,Q are the square invertible matrices of the appropriate size so that:

P.A.Q = J. Where J is the matrix of the form (having MATRIX's shape):

| I\_p O | where p is MATRIX's rank and I\_p Identity matrix of size p.

| 0 0 |

Therfore, we perform permuations of rows and columns in refAref (same changes

are applied to Q1 in order to get final Q matrix)

NOTE: P IS NOT RETURNED BECAUSE WE DO NOT NEED IT TO SOLVE H.G' = 0

P IS INVERTIBLE, WE GET SIMPLY RID OF IT.

Then

solves: inv(P).J.inv(Q).G' = 0 (1) where inv(P) = P^(-1) and

P.H.Q = J. Help(PJQ) for more info.

Let Y = inv(Q).G', equation becomes J.Y = 0 (2) whilst:

J = | I\_p O | where p is H's rank and I\_p Identity matrix of size p.

| 0 0 |

Knowing that G must have full rank, a solution of (2) is Y = | 0 | Where k = n-p.

| I-k |

Because of rank-nullity theorem.

-----------------

parameters:

H: Parity check matrix.

use\_sparse: (optional, default True): use scipy.sparse format to speed up calculations

---------------

returns:

tG: Transposed Coding Matrix.

"""

H = np.copy(MATRIX)

m,n = H.shape

if m > n:

raise ValueError('MATRIX must have more rows than columns (a parity check matrix)')

if n > 500 and use\_sparse:

sparse = 1

else:

sparse = 0

##### DOUBLE GAUSS-JORDAN:

Href\_colonnes,tQ = GaussJordan(np.transpose(H),1)

Href\_diag = GaussJordan(np.transpose(Href\_colonnes))

Q=np.transpose(tQ)

k = n - sum(Href\_diag.reshape(m\*n))

Y = np.zeros(shape=(n,k)).astype(int)

Y[n-k:,:] = np.identity(k)

if sparse:

Q = csr\_matrix(Q)

Y = csr\_matrix(Y)

tG = BinaryProduct(Q,Y)

return tG