LINEAR ALGEBRA ASSIGNMENT

UNIT 1

1.	Implementation of Basic Cryptographic techniques to appreciate Inverse functionality of Matrix.
Code	(see code below)

```
#Implementation of Hill Cipher!

#Notation Used:
#K = Matrix which is our 'Secret Key'
#P = Vector of plaintext (that has been mapped to numbers)
#C = Vector of Ciphered text (in numbers)

import numpy as np
from egcd import egcd

alphabet = "abcdefghijklmnopqrstuvwxyz "
#zip function - maps the first element in the alphabet to the first element of
the range len of alphabet
#here a is mapped to 0,b mapped to 1, and so on...
letter_to_index = dict(zip(alphabet, range(len(alphabet))))   #mapping
alphabets to numbers
index_to_letter = dict(zip(range(len(alphabet)), alphabet))   #mapping numbers
back to alphabets

def matrix_mod_inv(matrix, modulus):
    #To find the matrix modulus inverse by
    det = int(np.round(np.linalg.det(matrix)))  # Finding determinant
    det_inv = egcd(det, modulus)[1] % modulus  # Finding determinant value in a
specific modulus ie.length of alphabet
    matrix_modulus_inv = (
        det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
#Take that det_inv times the det*inverted matrix (this will then be the
adjoint) in mod 26
    )
    return matrix_modulus_inv     #since inverse = adjoint*1/determinant

#To Encrypt message
def encrypt(message, K):
    encrypted = ""        #To store the encrypted text
    message_in_numbers = []

    #Converting message to numbers
    for letter in message:
        message_in_numbers.append(letter_to_index[letter])

    #Split into the size of Matrix K so we can multiply the matrix
    split_P = [
        message_in_numbers[i : i + int(K.shape[0])]
        for i in range(0, len(message_in_numbers), int(K.shape[0]))
    ]
```

```
    #Iterate through each partial message and encrypt using K*P (mod 26)
    for P in split_P:
        P = np.transpose(np.asarray(P))[:, np.newaxis]

        while P.shape[0] != K.shape[0]:   #if this is not true we cannot do
matrix multiplication as multiplication ca happen only between p*q and q*r
            P = np.append(P, letter_to_index[" "])[:, np.newaxis]

        numbers = np.dot(K, P) % len(alphabet)   #multiplying K and P and
moding
        n = numbers.shape[0]  # length of encrypted message (in numbers)

        # Maping back to get encrypted text
        for idx in range(n):
            number = int(numbers[idx, 0])
            encrypted += index_to_letter[number]

    return encrypted

#To decrypt message
def decrypt(cipher, Kinv):
    decrypted = ""       #To store the decrypted text
    cipher_in_numbers = []

    #converting ciphered text into numbers
    for letter in cipher:
        cipher_in_numbers.append(letter_to_index[letter])

    #Split it into the sixe of the matrix inv(K) so we can multiply the matrix
    split_C = [
        cipher_in_numbers[i : i + int(Kinv.shape[0])]
        for i in range(0, len(cipher_in_numbers), int(Kinv.shape[0]))
    ]

    #Iterate through each partial ciphertext and decryot using inv(K)*C (mod
26)
    for C in split_C:
        C = np.transpose(np.asarray(C))[:, np.newaxis]
        numbers = np.dot(Kinv, C) % len(alphabet)
        n = numbers.shape[0]

    #Maping back the numbers to decrypted text
        for idx in range(n):
            number = int(numbers[idx, 0])
            decrypted += index_to_letter[number]

    return decrypted

def main():
    message = 'maths is fun'

    K = np.matrix([[3,10,20],[20,19,17], [23,78,17]])
```

	<pre> Kinv = matrix_mod_inv(K, len(alphabet))
	
	    encrypted_message = encrypt(message, K)
	    decrypted_message = decrypt(encrypted_message, Kinv)
	
	    print("Original message: " + message)
	    print("Encrypted message: " + encrypted_message)
	    print("Decrypted message: " + decrypted_message)
	
	main()</pre>

Output	<pre>[Running] python -u "d:\PES\sem4\LA\hill_cipher.py"
	Original message: maths is fun
	Encrypted message: lxftgjw fq g
	Decrypted message: maths is fun
	
	[Done] exited with code=0 in 0.299 seconds</pre>

code	**Mathlab code**
	
	encrypt.m
	
	function new_code = encrypt
	text = input('input text…\n','s')
	msg = double(text);
	msg = reshape(msg,2,length(text)/2);
	k= [1 0;0 3];
	code = mod(k*(msg-65) ,26) +65;
	new_code = reshape(code,1,length(text));
	new_code = char(new_code);
	
	decrypt.m
	function decrypt(text)
	k = [1 0;0 3];
	k = inv(k);
	k(2,2) +26/3;
	k(2,2) = ans;
	msg=double(text);
	q=reshape(msg,2,length(text)/2);
	code = mod(k*(q - 65),26) +65;
	new_code = reshape(code,1,length(text));
	new_code = char(new_code)

Output	<pre>>> encrypt
	input text...
	CRYPTOGRAPHY
	
	text =
	
	    'CRYPTOGRAPHY'
	
	ans =
	
	    'CZYTTQGZATHU'
	
	>> decrypt(ans)
	
	new_code =
	
	    'CRYPTOGRAPHY'</pre>

****