Course Code: CSE 4632

Course Name: Digital Signal Processing Lab

Lab No: 1

Name: Md Farhan Ishmam

ID: 180041120

Lab Group: P

Task-1

Explanation: I am importing the libraries required for all the tasks

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
```

Task-1a

Explanation: In this task, to create a row vector x of 5 equally spaced elements between 2 and 3, I am using `linspace()` to generate the required vector. `linspace()` divides a line segement into a finite number of points, and returns those points as a numpy array.

Then I am adding 1 to the second element by simply using the index of the array. Just like C and C++, arrays in Python start at 0. To access the second element, I am taking 1 as the index number of the array x.

```
1 x = np.linspace(2,3,5)
2 x[1] += 1
3 print(x)
4 print(x[1])
```

```
[2.   3.25 2.5  2.75 3.  ]
3.25
```

Task-1b

Explanation: In this task, to create a row vector y of the same dimension as x, I am using `arange()`. The number of elements have to be equal to the successive even integers starting with 4. `arange()`

has a starting point, an ending point and length of the step taken from starting point to reach the ending point. I am mathematically calculating the endpoint by accessing the `shape` of the vector x, and taking the first element of the returned tuple.

```
end_point = 4 + 2 * x.shape[0]
y = np.arange(4, end_point, 2)
print(y)
```

```
[ 4  6  8 10 12]
```

Task-1c

Explanation: In this task, I need to create the matrix A, whose first row is equal to x, second row is a line of ones, and third row is equal to y. I am combining the arrays in a list to from the matrix. Then I am converting that python list to a numpy array using the `np.array()`. For the ones row, I am using `np.ones()` to generate a vectors of ones.

```
A = np.array([x,np.ones(5),y])
print(A)
```

```
[[ 2.    3.25  2.5   2.75  3.  ]
 [ 1.    1.    1.    1.    1.  ]
 [ 4.    6.    8.   10.   12.  ]]
```

Task-1d

Explanation: In this task, I am defining a row vector z, whose elements are equal to the mean value of the columns of A by using the `np.mean()` function. `np.mean()` calcuates the mean of a numpy array where the `axis = 0` indicates column-wise mean of the matrix.

```
z = np.mean(A, axis = 0)
print(z)
```

```
[2.33333333 3.41666667 3.83333333 4.58333333 5.33333333]
```

Task-1e

Explanation: In this task, I am defining a column vector zz, whose elements are the sum of the elements in each rows of A by using the `np.sum()` function. This function sums along a row or column of a numpy array where `axis = 1` indicates row-wise summation.

```
zz = np.sum(A, axis = 1)
print(zz)
```

```
[13.5  5.  40. ]
```

Task-2

Explanation: In this task, I am creating two matrices using the `np.array()` function and passing a list of lists.

```
A = np.array([
              [1,2],
              [4,-1]
])
B = np.array([
              [4,-2],
              [-6,3]
])
print("The value of A is:")
print(A)
print("The value of B is:")
print(B)
```

```
The value of A is:
[[ 1  2]
 [ 4 -1]]
The value of B is:
[[ 4 -2]
 [-6  3]]
```

Task-2a

Explanation: In this task, I am adding and subtracting the matrices using the `+` and `-` operator. Numpy overloads the `+` and `-` operators for numpy arrays.

```
C1 = A+B
C2 = A-B

print("The value of C1 is:")
print(C1)
print("The value of C2 is:")
print(C2)
```

```
The value of C1 is:
[[ 5  0]
 [-2  2]]
```

```
The value of C2 is:
[[-3  4]
 [10 -4]]
```

Task-2b

Explanation: In this task, I am using the `np.dot()` function to do matrix multiplication.

```
D1 = np.dot(A,B)
D2 = np.dot(B,A)
print("The value of D1 is:")
print(D1)
print("The value of D2 is:")
print(D2)
```

```
The value of D1 is:
[[ -8   4]
 [ 22 -11]]
The value of D2 is:
[[ -4  10]
 [  6 -15]]
```

Task-2c

Explanation: In this task, I am using the `np.multiply()` function to perform element-wise multiplication. In order to reduce the matrix to its 4th root, I am using the `**` operator since numpy overloads this operator for numpy arrays.

I am adding 0j to convert the matrix B's data type to complex data type. This operation is done because there are negative numbers inside the fourth root, and that would result in complex values. Finally, I am using the `np.absolute()` function to find the magnitude of the complex number.

```
f = np.absolute(B + np.multiply(A, (B+0j)**(1/4)))
print(f)
```

```
[[5.41421356 1.71163166]
 [4.69798933 1.68392599]]
```

Task-2d

Explanation: In this task, I need to subtract from the second row, the first row multiplied by 4 for the matrix A. I am accessing the rows using array slicing based on the given conditions and then updating that particular row of the matrix A.

```
A[1,:] = A[1,:] - A[0,:]*4
```

```
print(A)
```

```
[[ 1  2]
 [ 0 -9]]
```

Task-3

Explanation: In this task, I am creating an array a based on the given condition and then summing over all the elements of the array. I am initializing the array with zeros. I am using a `for` loop to iteratively compute the values of each element. The `range()` function returns a list of numbers which are accessed by the loop. To use pi, I am using the constant declared in numpy as `np.pi`.

I am also iteratively dividing the factorial value or else the factorials of higher values result in a number too big to be converted to float. I am storing each element in the array a and also adding each calculated element to the sum variable initialized at 0.

```
a = np.zeros((101))     #Initializing the vector a
sum = 0                 #Initializing the sum
for n in range(101):
  num = ((-1)**n) * (np.pi**(2*n))
  for fac in range(2, (2*n + 1)): #Performing factorial and dividing it
    num = num/fac
  a[n] = num
  sum += num
print(a)
print("The sum is:", sum)
```

```
[ 1.00000000e+000 -4.93480220e+000  4.05871213e+000 -1.33526277e+000
  2.35330630e-001 -2.58068914e-002  1.92957431e-003 -1.04638105e-004
  4.30306959e-006 -1.38789525e-007  3.60473080e-009 -7.70070713e-011
  1.37686473e-012 -2.09063234e-014  2.72932726e-016 -3.09625062e-018
  3.08052104e-020 -2.70976150e-022  2.12256143e-024 -1.48996029e-026
  9.42648628e-029 -5.40276948e-031  2.81835082e-033 -1.34376848e-035
  5.87875148e-038 -2.36820210e-040  8.81343058e-043 -3.03931073e-045
  9.73921902e-048 -2.90750874e-050  8.10620369e-053 -2.11541575e-055
  5.17815391e-058 -1.19128976e-060  2.58067573e-063 -5.27334339e-066
  1.01811059e-068 -1.86011639e-071  3.22080928e-074 -5.29272618e-077
  8.26536607e-080 -1.22818268e-082  1.73862265e-085 -2.34740326e-088
  3.02611566e-091 -3.72865973e-094  4.39565175e-097 -4.96263371e-100
  5.37052977e-103 -5.57595248e-106  5.55883284e-109 -5.32551748e-112
  4.90671684e-115 -4.35106506e-118  3.71610340e-121 -3.05892164e-124
  2.42843842e-127 -1.86055943e-130  1.37653564e-133 -9.84054922e-137
  6.80128347e-140 -4.54721428e-143  2.94251286e-146 -1.84390082e-149
  1.11949875e-152 -6.58855680e-156  3.76049324e-159 -2.08251491e-162
  1.11947703e-165 -5.84406825e-169  2.96395898e-172 -1.46104798e-175
  7.00270276e-179 -3.26470978e-182  1.48103484e-185 -6.54014675e-189
  2.81233274e-192 -1.17802443e-195  4.80836853e-199 -1.91311357e-202
  7.42204171e-206 -2.80855055e-209  1.03693262e-212 -3.73644205e-216
  1.31441420e-219 -4.51540140e-223  1.51520555e-226 -4.96793546e-230
```

```
  1.59193369e-233 -4.98690909e-237  1.52758597e-240 -4.57673158e-244
  1.34148640e-247 -3.84770128e-251  1.08019369e-254 -2.96883443e-258
  7.99007999e-262 -2.10616230e-265  5.43877254e-269 -1.37616094e-272
  3.41260403e-276]
The sum is: -1.0
```

Task-4

Task-4a

Explanation: In this task, I am setting the negative values of x to zero using `clip()`. `clip()` keeps values within particular threshold where the `min = 0` attribute means that any value lesser than zero will be changed to zero.

```
x = np.array([7,6,1,2,0,-1,4,3,-2,0])
x = x.clip(min = 0)
print(x)
```

```
[7 6 1 2 0 0 4 3 0 0]
```

Task-4b

Explanation: In this task, I am extracting the values of x greater than 3 in a vector y by putting the condition within the array x's index. This will return a numpy array meeting the condition.

```
y = x[x>3]
print(y)
```

```
[7 6 4]
```

Task-4c

Explanation: In this task, I am adding 3 to the values of x that are even using the `np.where()` function which basically performs an if-else statement on a numpy array.

```
x = np.where(x%2 == 0,x+3, x )
print(x)
```

```
[7 9 1 5 3 3 7 3 3 3]
```

Task-4d

Explanation: In this task, I am settting the values of x that are less than the mean to zero using the `np.mean()` nested in `np.where()`

```
x = np.where(x<np.mean(x),0, x)
print(x)
```

```
[7 9 0 5 0 0 7 0 0 0]
```

Task-4e

Explanation: In this task, I am setting the values of x that are greater than the mean to their difference with the mean using the `np.mean()` nested in `np.where()` If condition is met, I am setting the number to difference of the number and mean or else I am keeping the number unchanged.

```
x = np.where(x>np.mean(x),x-np.mean(x), x)
print(x)
```

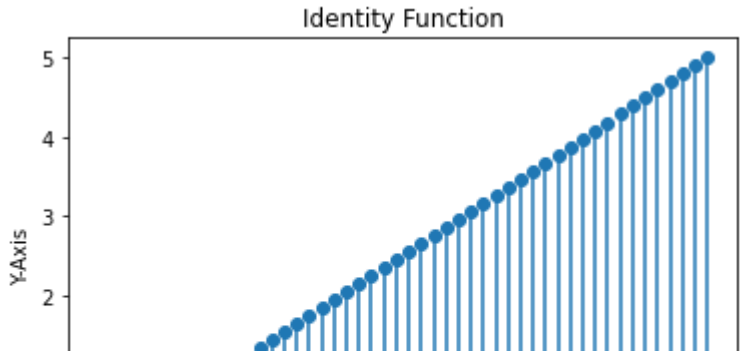
```
[4.2 6.2 0.  2.2 0.  0.  4.2 0.  0.  0. ]
```

Task-5

Explanation: In this task, I am using matplotlib to plot the graphs. A set of points for the x-axis is taken using `np.linspace()` and the corresponding function for the y-axis values is calculated. Finally, I am using `plt.stem()` taking the x and y axis points as arguments. I am, then, changing the titles and labels accordingly. I am plotting the graph with 50 points, and this can be changed by changing the third argument passed in the `np.linspace()` function.

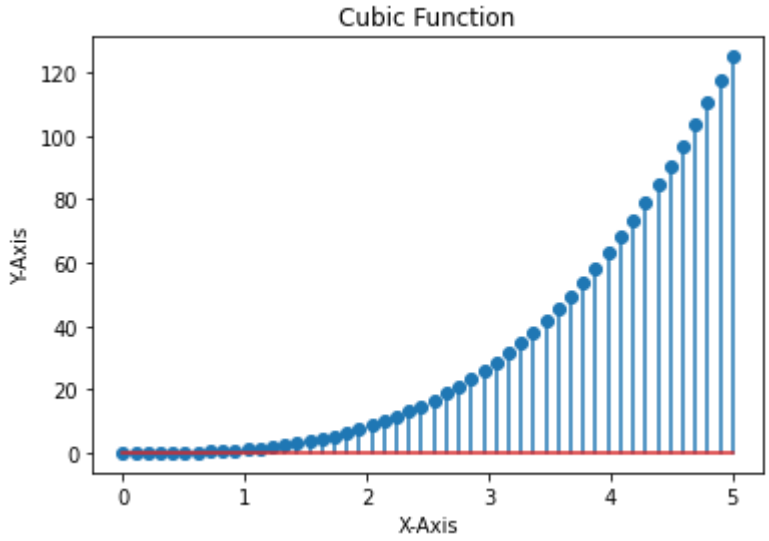
```
x = np.linspace(0,5, 50)
y = x
plt.stem(x, y)
plt.title("Identity Function")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: In Matplotl
  This is separate from the ipykernel package so we can avoid doing imports until
Text(0, 0.5, 'Y-Axis')
```



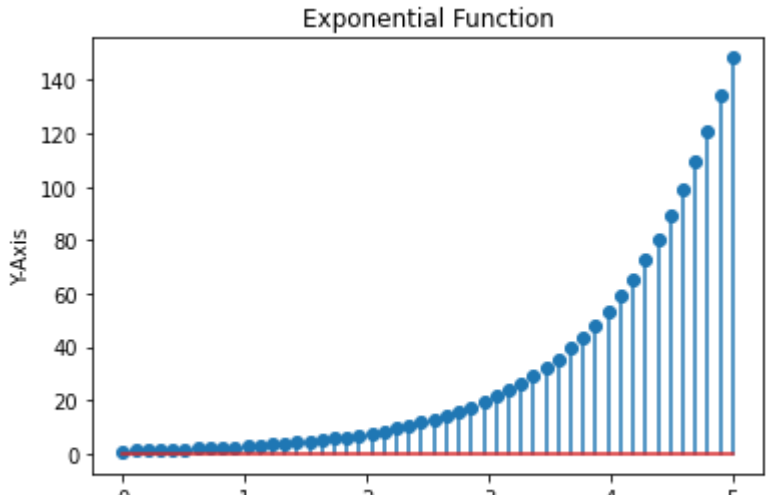

```
x = np.linspace(0,5, 50)
y = x**3
plt.stem(x, y)
plt.title("Cubic Function")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: In Matplotl
  This is separate from the ipykernel package so we can avoid doing imports until
Text(0, 0.5, 'Y-Axis')
```



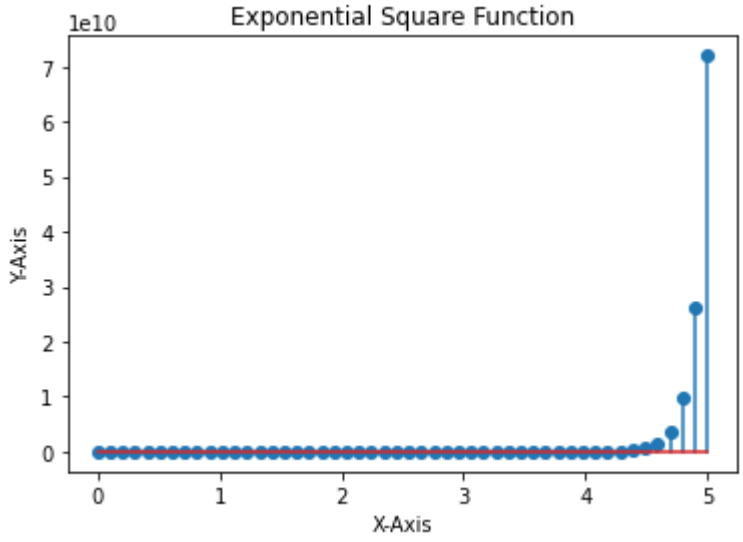

```
x = np.linspace(0,5, 50)
y = np.exp(x)
plt.stem(x, y)
plt.title("Exponential Function")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: In Matplotl
  This is separate from the ipykernel package so we can avoid doing imports until
Text(0, 0.5, 'Y-Axis')
```




```
x = np.linspace(0,5, 50)
y = np.exp(x**2)
plt.stem(x, y)
plt.title("Exponential Square Function")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: In Matplotl
  This is separate from the ipykernel package so we can avoid doing imports until
Text(0, 0.5, 'Y-Axis')
```



Task-6

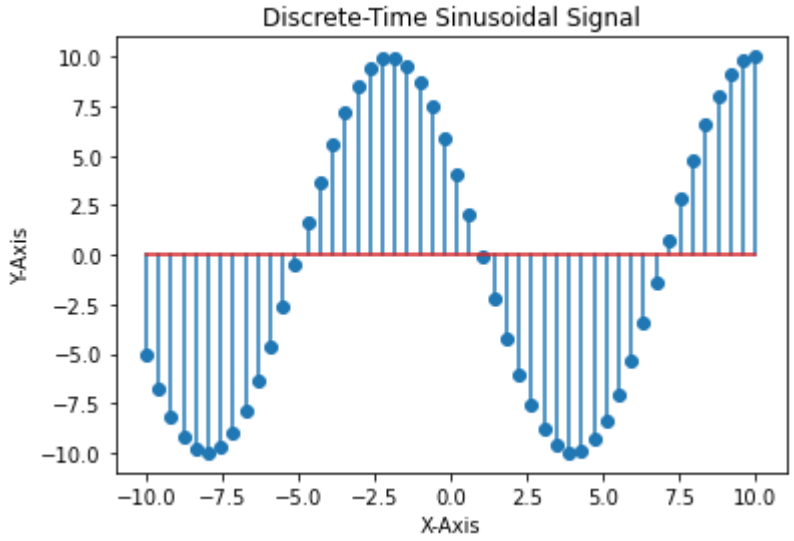
Explanation: In this task, I am using matplotlib to plot the graphs. This task is similar to task-5. I am declaring variables A, theta, and omega and directly using them in the equation of y. Similar to task-

5, I am using `np.linspace()` to generate values of n and using `plt.stem()` to plot the graph.

```
n = np.linspace(-10,10, 50)
omega = np.pi/6
theta = np.pi/3
A = 10
y = A*np.cos(omega*n + theta)
plt.stem(n, y)
plt.title("Discrete-Time Sinusoidal Signal")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotl

Text(0, 0.5, 'Y-Axis')
```



Task-7

Explanation: In this task, I am creating a python function that takes two values of t (t1 and t2) and plots the population in that range. Within the function, I am taking the necessary x-axis points with the corresponding y function to generate the y-axis values. Finally, I am plotting the points with a title and labels using `plt.plot()`

Task-7a

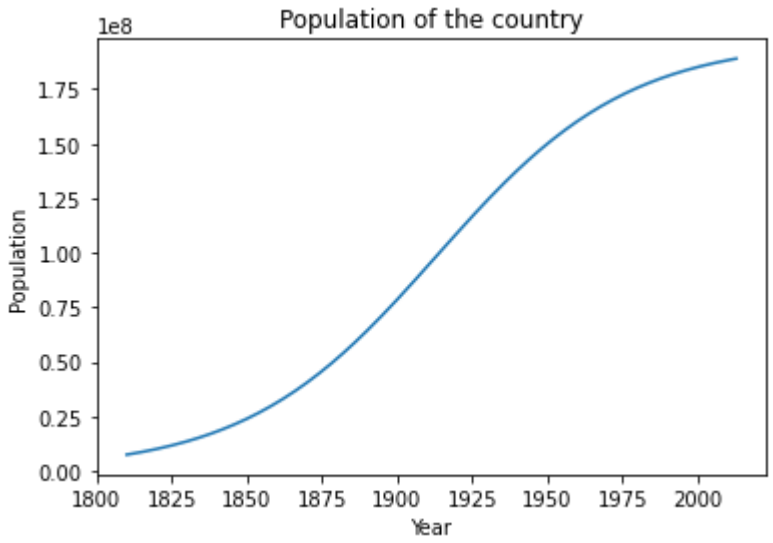
```
def population_plot(t1, t2):
  t = np.arange(t1,t2+1,1)
  p = 197273000/(1 + np.exp(-0.0313*(t - 1913.26)))
  plt.plot(t, p)
  plt.title("Population of the country")
```

```
6   plt.xlabel("Year")
7   plt.ylabel("Population")
8   return
```

Task-7b

Explanation: In this task, I am verifying the function by passing 1810 and 2013 as arguments for t1 and t2 respectively.

```
1 population_plot(1810,2013)
```



Task-7c

Explanation: In this task, I am predicting the population in the year 2021 by using the given formula to generate the y-values with the year set as 2021.

```
1 year = 2021
2 p = 197273000/(1 + np.exp(-0.0313*(year - 1913.26)))
3 print("Population in 2021 is", p)
```

```
Population in 2021 is 190728728.107411
```

Task-8

Explanation: In this task, I am creating a 16x16 matrix as given in the question using `np.diagflat()` to create diagonals. The second argument `1` means the diagonal above the main diagonal, `0` means the diagonal and `-1` means the diagonal below the main diagonal. I am using `np.ones()` to create an array of ones and `np.repeat()` to create an array of -2. I am passing these

to the `np.diagflat()` function along with the proper second argument. I am, then, adding the diagonal matrices and changing the corner elements to 1. Finally, I am coverting the array of floating point data type to an array of integer data type using `.astype(int)`

```
d1 = np.diagflat(np.ones(15), 1)
d2 = np.diagflat(np.repeat(-2,16), 0)
d3 = np.diagflat(np.ones(15), -1)
D = d1 + d3 + d2
D[15,0] = 1
D[0,15] = 1
D = D.astype(int)
print(D)
```

```
[[-2  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1]
 [ 1 -2  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  1 -2  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  1 -2  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1 -2  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  1 -2  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  1 -2  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1 -2  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1 -2  1  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1 -2  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1 -2  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  1 -2  1  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1 -2  1  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1 -2  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1 -2  1]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  1 -2]]
```

✓ 0s completed at 11:00 AM