Advanced Digital Signal Processing (ADSP) Lab - Python Lab Manual

Course Code: EEE G613 **IC:** Dr. Rajesh Kumar Tripathy **Research Scholar:** Shaswati Dash **Lab Technician:** Ramesh Pokanati

---

Experiment No. - 2

1. Generate two random variables X_1, X_2 of length 10. Calculate the covariance matrix without using any built-in matlab commands and compare your results with the existing matlab command for covariance

Pseudocode:

- use length(x1), length(x2) command for finding out length of input sequence
- use mean(x1) and mean(x2) for finding out mean of two sequences
- find covariance of both the signals using for loop ranging 1:length(x1)
- verify the results using inbuilt covariance command.

Python Code:

```
#import libraries
import numpy as np
import matplotlib.pyplot as plt

X1 = np.random.rand(10) #random variable of length 10
X2 = np.random.rand(10) #random variable of length 10
```

```
len1 = len(X1) #length of X1
len2 = len(X2) #length of X2
avg1 = np.mean(X1)
avg2 = np.mean(X2)
S1 = X1-avg1
S2 = X2-avg2
C = np.vstack((S1, S2))
cov1 = np.dot(C, C.T) / (len1-1)
#
cov2 = np.cov(X1, X2)
print("Covariance Matrix w/o using built-in function:")
print(cov1)
print("Covariance matrix with using built-in function:")
print(cov2)
```

```
Covariance Matrix w/o using built-in function:
[[0.1049128  0.01130843]
 [0.01130843 0.07847073]]
Covariance matrix with using built-in function:
[[0.1049128  0.01130843]
 [0.01130843 0.07847073]]
```

2. a) Generate a Gaussian random vector of length 10 (10 random values) by getting mean and variance as inputs from the user

   b) Consider for example N = 2, the vector being x= [x1 x2]' x1 and x2 are Gaussian random variables. Take 100 samples for x_1 and x_2 from the distribution and calculate its covariance matrix (write your own matlab code do not use built-in function). Also for the above case calculate its *correlation coefficient* matrix (write your own matlab code do not use built-in function). Compare your answers with the built in matlab commands.

Pseudocode:

- get input from user
- generate a gaussian vector for the input sequences

- check the length of both the input sequences
- calculate coefficient of correlation using formula.

Python Code:

```
#Q2-part(a)
#import libraries
import numpy as np
import time

start_time = time.time()



# Function to generate Gaussian random vector
def generate_gaussian_vector(length, user_mean, user_variance):
  # Generate Gaussian random vector with user-specified mean and
variance
  gaussian_vector = np.random.normal(user_mean,
np.sqrt(user_variance),length)
  return gaussian_vector

# Function to calculate covariance matrix
def calculate_covariance_matrix(data):
  mean_vector = np.mean(data, axis=0)
  centered_data = data - mean_vector
  covariance_matrix = np.dot(centered_data.T, centered_data) /
(len(data) - 1)
  return covariance_matrix

# Function to calculate correlation coefficient matrix
def calculate_correlation_matrix(data):
  covariance_matrix = calculate_covariance_matrix(data)
  std_dev_vector = np.sqrt(np.diagonal(covariance_matrix))
  correlation_matrix = covariance_matrix /
np.outer(std_dev_vector,std_dev_vector)
  return correlation_matrix

# User inputs for Gaussian random vector
length = 10
user_mean = float(input("Enter the mean for the Gaussian random
vector: "))
user_variance = float(input("Enter the variance for the Gaussian
random vector:"))
```

```
Enter the mean for the Gaussian random vector: 2
Enter the variance for the Gaussian random vector:1

# Generate Gaussian random vector
gaussian_vector = generate_gaussian_vector(length, user_mean,
user_variance)
# Reshape vector for the case with N=2 (as in the example)
#The reshape function is used to change the shape of the Gaussian
vector. In this case, it transforms
#the 1D vector into a 2D array with two columns
gaussian_vector_reshape = gaussian_vector.reshape((length // 2, 2))
# Calculate covariance matrix and correlation coefficient matrix
manually
covariance_matrix_calculated =
calculate_covariance_matrix(gaussian_vector_reshape)
correlation_matrix_calculated =
calculate_correlation_matrix(gaussian_vector_reshape)
# Calculate covariance matrix and correlation coefficient matrix using
NumPy's built-in functions
covariance_matrix_builtin = np.cov(gaussian_vector_reshape,
rowvar=False)
correlation_matrix_builtin = np.corrcoef(gaussian_vector_reshape,
rowvar=False)
# Compare and print the results
print("\nGenerated Gaussian Random Vector:\n", gaussian_vector)
print("\nCalculated Covariance Matrix (Manual):\
n",covariance_matrix_calculated)
print("\nCalculated Covariance Matrix (NumPy's built-in):\
n",covariance_matrix_builtin)
print("\nCalculated Correlation Coefficient Matrix (Manual):\
n",correlation_matrix_calculated)
print("\nCalculated Correlation Coefficient Matrix (NumPy's built-
in):\n",correlation_matrix_builtin)


Generated Gaussian Random Vector:
 [-0.52996264  3.54321582  0.38072742  1.59081828  2.01253798
2.1164046
  3.78898459  1.58606165  1.32581229  1.47215652]

Calculated Covariance Matrix (Manual):
 [[ 2.71288261 -0.85958996]
 [-0.85958996  0.74835116]]

Calculated Covariance Matrix (NumPy's built-in):
 [[ 2.71288261 -0.85958996]
 [-0.85958996  0.74835116]]

Calculated Correlation Coefficient Matrix (Manual):
 [[ 1.         -0.60328581]
```

```
 [-0.60328581  1.        ]]

Calculated Correlation Coefficient Matrix (NumPy's built-in):
 [[ 1.         -0.60328581]
 [-0.60328581  1.        ]]
```

```
# Calculate errors
covariance_error = np.linalg.norm(covariance_matrix_calculated -
covariance_matrix_builtin)
correlation_error = np.linalg.norm(correlation_matrix_calculated -
correlation_matrix_builtin)
print(f"\nCovariance Matrix Error: {covariance_error}")
print(f"Correlation Coefficient Matrix Error: {correlation_error}")
```

```
Covariance Matrix Error: 0.0
Correlation Coefficient Matrix Error: 1.5700924586837752e-16
```

```
# Calculate elapsed runtime
elapsed_time = time.time() - start_time
print(f"\nElapsed Runtime: {elapsed_time:.4f} seconds\n")
```

```
Elapsed Runtime: 8.9496 seconds
```

3. Generate a sinusoidal signal x(k) of 100 samples. Calculate and plot the sample autocorrelation function $r_x(k)$ for |lag| < 100.

Pseudocode:

- Generate sinusoidal signal. ex: t= 1:1:100; f=0.1;
- Use autocorrelation function formula for calculation rx(k)

Python Code:

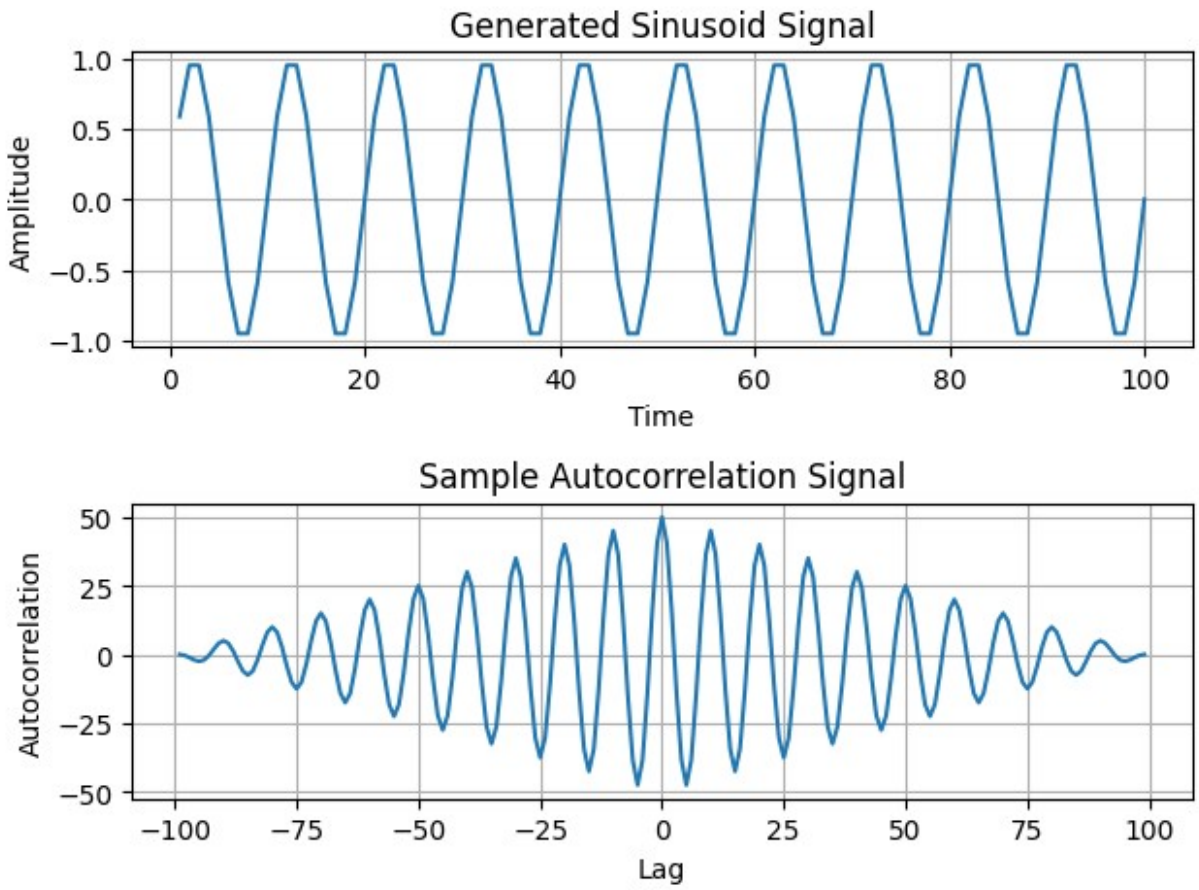
```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
import time

start_time = time.time()
```

```
t = np.arange(1, 101, 1)
f = 0.1
y = np.sin(2 * np.pi * f * t)

# Plot the generated sinusoid signal
plt.subplot(2, 1, 1)
plt.plot(t, y)
plt.title('Generated Sinusoid Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

# Calculate and plot the sample autocorrelation signal
rxx, lags = np.correlate(y, y, mode='full'), np.arange(-len(y) + 1,
len(y))
plt.subplot(2, 1, 2)
plt.plot(lags, rxx)
plt.title('Sample Autocorrelation Signal')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.grid(True)
# Adjust layout for better visualization
plt.tight_layout()
# Show the plots
plt.show()
# Calculate elapsed runtime
elapsed_time = time.time() - start_time
print(f"\nElapsed Runtime: {elapsed_time:.4f} seconds\n")
```




```
Elapsed Runtime: 0.4850 seconds
```