



FEBRUARY 15, 2024


# CS-963 COMPUTER NETWORKING-1

## MOHAMMED JAWED

ASSIGNMENT 2

MOHAMMED, JAWED

233560019  
Cybersecurity



**Step1: Finding the Path Loss Exponent (20 points)**

The purpose of this step is to find out the path loss exponent of an unknown environment. Use any programming language/tools to solve your problem. Describe the outcomes in a report while submitting.

- 1) First open the spreadsheet named HW2\_part1.csv; the spreadsheet consists of 13 RSSI (Signal strength) values from columns B-N in dBm, with different distances in meters (in column A). So, in the same location, the RSSI values are slightly different for different measurements.

To accomplish this, in Step 1, I utilized a Python script available in **Appendix A** of this assignment document.

I've designed the Python code to read data directly from the provided CSV file instead of manually inputting the distance and sample data.

Below is the formula incorporated into the Python script to convert distance (in meters) to distance in a logarithmic scale (base 10). I have also include Average RSSI Sample date row wise for refrence.

```
x_values.append(math.log(float(row[0]),10))
```

Result:

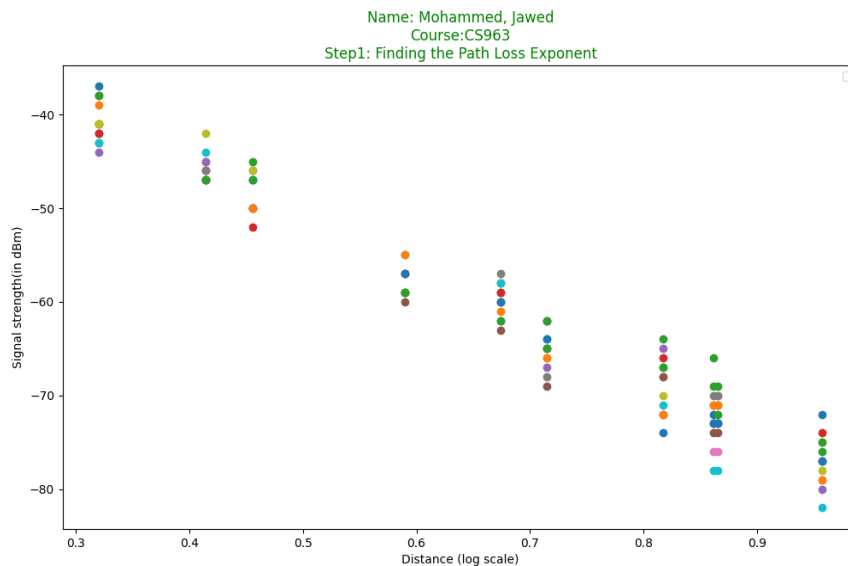
	Distance(In meter)	Distance(In logscale)		Average of RSSI Sample Data(In dBm)
Row 1	2.091	0.320354	Row 1	-40.7692
Row 2	2.594	0.41397	Row 2	-45.8462
Row 3	2.856	0.455758	Row 3	-48.2308
Row 4	4.721	0.674034	Row 4	-59.7692
Row 5	3.891	0.590061	Row 5	-58
Row 6	5.189	0.715084	Row 6	-65
Row 7	6.562	0.817036	Row 7	-68.9231
Row 8	7.267	0.861355	Row 8	-72.0769
Row 9	9.059	0.95708	Row 9	-77
Row 10	7.336	0.865459	Row 10	-72.3846

Provided Sample Data(For Reference):

Distance(In meter)		RSSI Sample Data												
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2.091	-42	-41	-43	-42	-44	-38	-41	-41	-41	-43	-37	-39	-38	
2.594	-47	-45	-47	-46	-45	-47	-46	-46	-42	-44	-47	-47	-47	
2.856	-50	-50	-45	-52	-50	-50	-47	-46	-46	-47	-47	-50	-47	
4.721	-59	-60	-60	-59	-62	-63	-58	-57	-58	-58	-60	-61	-62	
3.891	-59	-59	-57	-57	-59	-60	-59	-59	-59	-55	-57	-55	-59	
5.189	-62	-62	-62	-65	-67	-69	-65	-68	-66	-64	-64	-66	-65	
6.562	-67	-68	-64	-66	-65	-68	-72	-72	-70	-71	-74	-72	-67	
7.267	-72	-71	-69	-73	-73	-74	-76	-70	-71	-78	-73	-71	-66	
9.059	-72	-75	-75	-74	-80	-77	-79	-77	-78	-82	-77	-79	-76	
7.336	-71	-70	-69	-73	-73	-74	-76	-70	-71	-78	-73	-71	-72	

2) Plot all these points in a graph where the RSSI values are in y-axis (dBm), and the distances are in x-axis (in log scale)

Please find below a screenshot of the graph representing all sample data, with RSSI on the Y-axis and Distance (in logscale) on the X-axis.



From the graph, it is evident that as the distance increases, the signal strength decreases (indicated by negative values, signifying a loss in signal strength).

3) Draw a best fit straight line corresponding to this log-log plot. Find out the slope of this line, divide it by 10 and take the absolute value, which is your path loss exponent.

To continue with the graph plotted for the above question (2), the following steps were included in the Python script to obtain the desired results, including the best-fitted line and path in loss exponent. [Please refer to the screenshot of the graph also.]

Calculated the average signal strength of the provided RSSI data corresponding to the provided distance.

```
# Calculate the average of each row across all columns
average_y_values = np.mean(y_values, axis=0)
# Convert lists to numpy arrays for linear regression
X = np.array(x_values).reshape(-1, 1)
y = average_y_values # store average of data as Y-axis
```

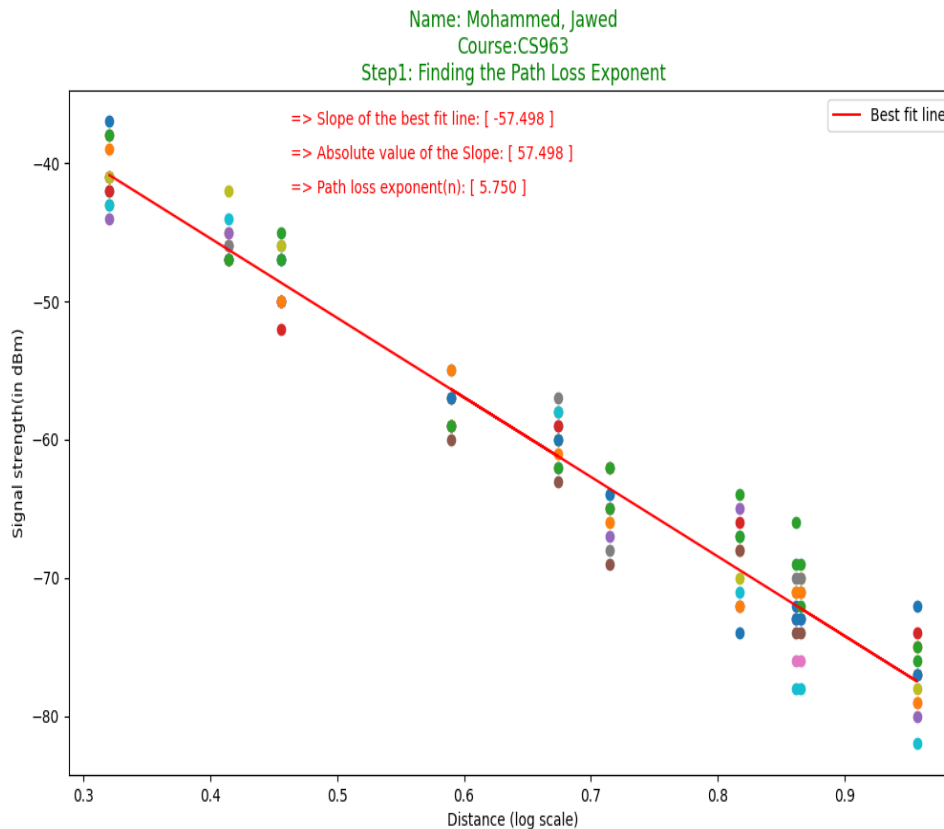
Now, through linear regression, I attempt to find the best-fitting line through the data points, where x represents the distance in log scale, and y represents the average of the RSSI sample data,

```
# Perform linear regression
reg = LinearRegression().fit(X, y)
# Calculate the slope of the line
slope = reg.coef_[0]
```

After that, to obtain the slope, I utilized the coefficient(s) of the independent variable (in our case, only one independent variable) from the best-fitted line.

- **Slope of the best fit line: -57.498**
- **Absolute value: 57.498**
- **Path loss exponents(n): 5.750**

Result is shown in below plotted graph: -



#### 4) Also find out the variance of these RSSI samples, w.r.t. the best fit line.

First, we will determine the residuals by subtracting the predicted values from the observed values. Then, we will calculate the variance of the residuals for each column (variable) by setting axis=0. This computes the variance along the rows (against the sample data collected for a given distance), which will provide us with the variance for each variable across all observations.

This is demonstrated below using the calculated data from the Python script.

```
# Calculate the residuals against avg. of samples RSSI data for given Distance
residuals = average_y_values - reg.predict(X)
# Calculate the variance of the residuals against best fitted line
variances = np.var(residuals, axis=0)
print(f"{variance:.3f}")
```

**Variance: 0.763**

**End of Q1**

### Step 2: Range Estimation (20 points)

The purpose of this step is to find out the distance/range from the path loss exponent that you have found in the last step. Use any programming language/tools to solve your problem. Describe the outcomes in a report while submitting.

$$Pr(d)[dBm] = Pt(d)[dBm] - [PL(d)]dB - 10n \log_{10}(d/d_0)$$

$$= Pr(d_0)[dBm] - 10n \log_{10}(d/d_0)$$

I have used a Python script to work on Step 2 of this assignment. The script is provided in Appendix B for reference. Through the Python script, I will be reading sample data directly from the CSV instead of hardcoding it into the script. All calculations are completed to three decimal points for convenience. Below is the sample data from HW2\_part2.csv.

RSSI Measurement Data

Distance(In meter)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	-27	-26	-28	-26	-27	-31	-26	-28	-26	-26	-25	-28	-26	-26	-25
1.296	-36	-38	-36	-38	-37	-36	-37	-38	-42	-42	-41	-40	-42	-41	-40
1.834	-33	-35	-35	-35	-36	-37	-35	-38	-37	-37	-38	-39	-37	-35	-33
2.9	-50	-51	-50	-50	-50	-47	-47	-48	-48	-48	-47	-45	-47	-45	-49
3.528	-60	-60	-58	-63	-65	-63	-61	-59	-60	-59	-60	-59	-58	-61	-62
5.347	-68	-71	-67	-67	-69	-70	-64	-65	-60	-64	-68	-71	-65	-68	-66

- Now use the obtained path loss exponent for estimating some distances, using the following formula (I have ignored the noise term). Use HW2\_part2.csv: column A is the distance in meters and columns B-P are the RSSI measurements at those distances. Assume  $d_0$  as 1 meter, and find  $P$ , ( $d_0$ ) by averaging columns B1-P1. Assume that column A is unknown, which you want to estimate based on the measurements of columns B-P.

Before proceeding to find the actual distance, I calculated the average of RSSI measurements across rows. As shown in the table screenshot below,

	Averg. measurements (In dBm)
Row 1 (B1- P1)	-26.7333
Row 2 (B2- P2)	-38.9333
Row 3 (B3- P3)	-36
Row 4 (B4- P4)	-48.1333
Row 5 (B5- P5)	-60.5333
Row 6 (B6- P6)	-66.8667

Now, considering the average value (B1-P1) of -26.733, I calculated the actual distance based on the formula below (calculated using a Python script):

```
for i in range(1,6):  
    exponent=(pd0-average_y_values[i])/(10*n)  
    actual_distance.append(float('{:.3f}'.format(pow(10, exponent))))
```

n => consider from Step1 => 5.750

Actual Distance calculated and formulated:

	Actual Distance (In meter)
Row 2 (B2- P2)	1.63
Row 3 (B3- P3)	1.449
Row 4 (B4- P4)	2.356
Row 5 (B5- P5)	3.871
Row 6 (B6- P6)	4.989

- 2) However, due to the noise there will be some errors in range/distance estimation. So, calculate the distance error by comparing with the actual distance. Repeat this experiment for 5 different distances (rows 2-6) that are given in the spreadsheet, and report the average error,

Results are displayed below in tabular format (rows 2-6, as row 1 is used as a reference). These results were calculated using a Python script.

**Average error: -0.122**, The negative average error indicates that, the estimated distances (Recorded/provided) are slightly underestimated compared to the actual distances.

	Recorded Distance	Actual Distance	Error
Row 2	1.296	1.63	0.334
Row 3	1.834	1.449	-0.385
Row 4	2.9	2.356	-0.544
Row 5	3.528	3.871	0.343
Row 6	5.347	4.989	-0.358
Average Error			-0.122

END OF Q2

## APPENDIX – A

```
import csv
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from tabulate import tabulate

# Path to the CSV file
csv_file = "HW2_part1.csv"

# Lists to store values from the CSV file
x_values = [] # For values from the first column after conversion to log scale
y_values = [[] for _ in range(13)] # For values from the rest of the columns
x_values_original=[]

# Open the CSV file in read mode
with open(csv_file, mode='r', newline='') as file:
    # Create a CSV reader object
    csv_reader = csv.reader(file)

    # Iterate over each row in the CSV file
    for row in csv_reader:
        # Check if row is not empty and the first column value is numeric
        if row and row[0].replace('.', '', 1).isdigit(): # Check if the value is numeric
            # Convert the value from the first column to log scale and append it to the list
            x_values.append(math.log(float(row[0]),10))
            x_values_original.append(row[0])

            # Append values from the rest of the columns to the corresponding lists
            for i in range(1, 14):
                y_values[i-1].append(float(row[i]))

table_x_axis_values=[]
for i,xaxis in enumerate(x_values):
    table_x_axis_values.append([f"Row {i+1}", x_values_original[i] , x_values[i]])

# Print the distance in tabular format with logscale
print(tabulate(table_x_axis_values, headers=[ "Distance(In meter)", "Distance(In logscale)" ], tablefmt="grid"))

# Calculate the average of each row across all columns
average_y_values = np.mean(y_values, axis=0)

# Convert lists to numpy arrays for linear regression
X = np.array(x_values).reshape(-1, 1)
```

```
y = average_y_values # store average of data as Y-axis

#Print Average Sample data
table_y_axis_avg_values=[]
for i,yaxisavg in enumerate(average_y_values):
    table_y_axis_avg_values.append([f"Row {i+1}", yaxisavg])

print(tabulate(table_y_axis_avg_values, headers=[ "Average of RSSI Sample Data(In dBm)"], tablefmt="grid"))
#print("Values of the first column in y_values:")
#print(y_values[0])

# Perform linear regression
reg = LinearRegression().fit(X, y)

# Calculate the slope of the line
slope = reg.coef_[0]

# Plotting the line graph
plt.figure(figsize=(12, 7))

# Plotting each column of y_values against x_values
for i, column_values in enumerate(y_values):
    #print(f"Column {i+2} values:", column_values)
    plt.scatter(x_values, column_values)

plt.plot(x_values, reg.predict(X), color='red', label='Best fit line')

# Adding labels and title
plt.xlabel("Distance (log scale)")
plt.ylabel("Signal strength(in dBm)")
plt.title('Name: Mohammed, Jawed\nCourse:CS963\nStep1: Finding the Path Loss Exponent',color='green')

# Adding legend
plt.legend()

# Calculate margins for x-axis and y-axis
x_margin = (max(x_values) - min(x_values)) * 0.05 # 5% margin
y_margin = (max([max(column_values) for column_values in y_values]) - min([min(column_values) for column_values in y_values])) * 0.05 # 5% margin

# Adjusting x-axis and y-axis limits with margins
plt.xlim(min(x_values) - x_margin, max(x_values) + x_margin)
```



```
plt.ylim(min([min(column_values) for column_values in y_values]) - y_margin, max([max(column_values) for column_values in y_values]) + y_margin)

# Calculate the absolute value of the slope divided by 10
slope_divided_by_10 = abs(slope) / 10

# Print values on the graph with three decimal places
plt.text(0.25, 0.95, f"=> Slope of the best fit line: [ {slope:.3f} ]", transform=plt.gca().transAxes, color='red')
plt.text(0.25, 0.90, f"=> Absolute value of the Slope: [ {abs(slope):.3f} ]", transform=plt.gca().transAxes, color='red')
plt.text(0.25, 0.85, f"=> Path loss exponent(n): [ {slope_divided_by_10:.3f} ]", transform=plt.gca().transAxes, color='red')

# Display the plot

# Calculate the residuals for each column of RSSI samples
residuals = average_y_values - reg.predict(X)

# Calculate the variance of the residuals for each column
variance = np.var(residuals, axis=0)
print(f"{variance:.3f}")

# Plotting each column of y_values against x_values
for i, avg_values in enumerate(average_y_values):
    print(f"Column {i+2} values:", avg_values)

# Show the Graph
plt.show()
```

## APPENDIX -B

```
import csv
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from tabulate import tabulate

# Path to the CSV file
csv_file = "HW2_part2.csv"

# Lists to store values from the CSV file
x_values = [] # For values from the first column after conversion to log scale
y_values = [[] for _ in range(15)] # For values from the rest of the columns
n=5.750 #Calculated from previous Question -1
# Open the CSV file in read mode
with open(csv_file, mode='r', newline='') as file:
    # Create a CSV reader object
    csv_reader = csv.reader(file)

    # Iterate over each row in the CSV file
    for row in csv_reader:
        # Check if row is not empty and the first column value is numeric
        if row and row[0].replace('.', '', 1).isdigit(): # Check if the value is numeric
            # Get the X-axis values which is distance
            x_values.append(row[0])
            # Append values from the rest of the columns to the corresponding lists
            for i in range(1, 16):
                y_values[i-1].append(float(row[i]))
# Print the y_values
print("Distance:", x_values)
# Calculate the average of each row across all columns
average_y_values = np.mean(y_values, axis=0)
y = average_y_values # store average of data as Y-axis
#Print Average Sample data
table_y_axis_avg_values=[]
for i,yaxisavg in enumerate(average_y_values):
    table_y_axis_avg_values.append([f"Row {i+1} \n (B{i+1}- P{i+1})", yaxisavg])

print(tabulate(table_y_axis_avg_values, headers=[ "Averg. measurements\n (ln dBm)" ], tablefmt="grid"))

#Calculate actual distance
pd0= average_y_values[0]
```

```
actual_distance = []
table_actual_distance=[]
for i in range(1,6):
    exponent=(pd0-average_y_values[i])/(10*n)
    actual_distance.append(float('{:.3f}'.format(pow(10, exponent))))
    table_actual_distance.append([f"Row {i+1} \n (B{i+1}- P{i+1})", float('{:.3f}'.format(pow(10, exponent)))]

print(tabulate(table_actual_distance, headers=[ "Actual Distance \n (In meter)"], tablefmt="grid"))

#print("Actual Distance:",actual_distance)

error_distance = []
averg_error_distance = []
table_error_distance=[]
table_error_average=[]
for i in range(1, 6):
    # Convert x_values[i] to a float
    x_value_float = float(x_values[i])
    actual_distance_float=actual_distance[i-1]
    error=float('{:.3f}'.format(actual_distance_float - x_value_float))
    error_distance.append(error)
    table_error_distance.append([f"Row {i+1}", x_value_float,actual_distance_float,error])

average_error=np.mean(error_distance)
table_error_average.append([f"Average Error",average_error])
# Print the distance in tabular format with logscale
print(tabulate(table_error_distance, headers=[ "Recorded Distance", "Actual Distance", "Error"],
tablefmt="grid"))

print(tabulate(table_error_average, tablefmt="grid"))
```