# Question 6

Design and create a computer program that produces an arbitrary number of samples of propagation pathloss using a d^n pathloss model with lognormal shadowing. Your program is a radio propagation simulator, and should use, as inputs, the T-R separation, frequency, the pathloss exponent, the standard deviation of the log-normal shadowing, the close-in-reference distance, and the number of desired predicted samples. Your program should provide a check that insures that the input T-R separation is equal to or exceeds the specified input close-in-reference distance, and should provide a graphical output of the produced samples as a function of pathloss and distance ( this is called a scatter plot).

Verify the accuracy of your computer program by running it for 50 samples at each of 5 different T-R separation distances (a total of 250 predicted pathloss values), and determine the best fit pathloss exponent and the standard deviation about the mean pathloss exponent of the predicted data using the techniques as described in example in the class. Draw the best fit mean pathloss model on the scatter plot to illustrate the fit of the model to the predicted values. You will know your simulator is working if the best fit pathloss model and the standard deviation for your simulated data is equal to the parameters you specified as inputs to your simulators.

## Code:

### Path Loss vs. Distance:

def pathloss\_model(distance, n, sigma, f, d0):

    # Check if the T-R separation is less than d0

    if np.any(distance < d0):

        raise ValueError("T-R separation must be greater than or equal to d0.")

    lambda\_=3e8/f

    # Calculate the pathloss without shadowing

    pathloss = 10\*n \* np.log10(distance / d0) + 20 \* np.log10(4 \* np.pi \* d0 / lambda\_ )

    # Add lognormal shadowing

    shadowing = np.random.normal(0, sigma)

    return pathloss + shadowing

distance\_values = np.array([100, 200, 300, 400, 500])

carrier\_frequency = 2.4e9

reference\_distance = 1

path\_loss\_exponent = 4

num\_samples = 50

shadowing\_stddev = 8

speed\_of\_light = 3e8 / carrier\_frequency

path\_loss\_matrix = np.zeros((len(distance\_values), num\_samples))

reference\_path\_loss = 20 \* np.log10(4 \* np.pi \* reference\_distance / speed\_of\_light)

for i, distance in enumerate(distance\_values):

    for j in range(num\_samples):

        path\_loss\_matrix[i, j] = pathloss\_model(distance, path\_loss\_exponent, shadowing\_stddev, carrier\_frequency, reference\_distance)

plt.figure(figsize=(8, 6))

plt.scatter(np.tile(distance\_values, 50), path\_loss\_matrix, s=10, alpha=0.5, label="Individual Measurements")

plt.plot(distance\_values, np.mean(path\_loss\_matrix, axis=1), marker="o", markersize=10, linestyle="-", color="red", label="Mean Path Loss")

plt.xlabel("Distance [meters]", fontsize=12)

plt.ylabel("Path Loss [dB]", fontsize=12)

plt.title("Path Loss vs. Distance", fontsize=16)

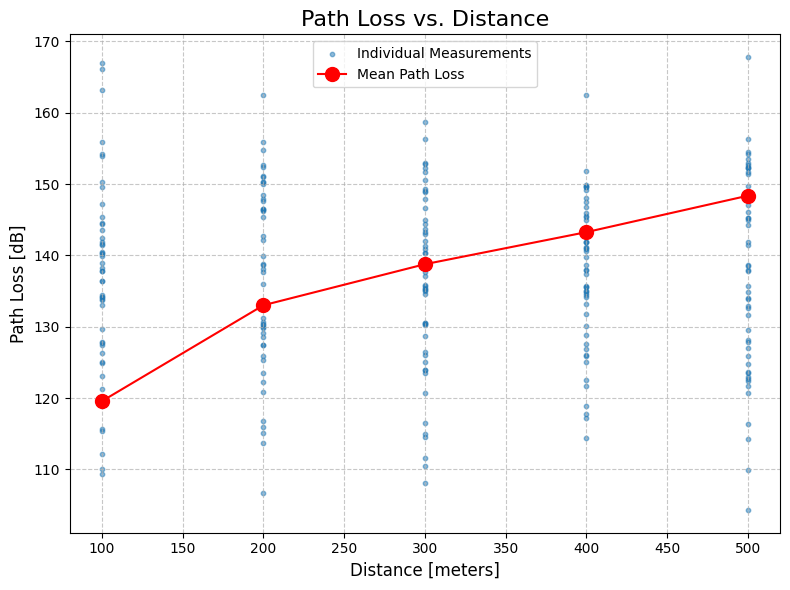
plt.grid(True, linestyle='--', alpha=0.7)

plt.legend()

plt.tight\_layout()

plt.show()

#### Output



### Estimating alpha

def calculate\_path\_loss\_estimate(

    distances: np.ndarray,

    reference\_path\_loss: float,

    reference\_distance: float,

    path\_loss\_exponent: float,

):

    return reference\_path\_loss + 10 \* path\_loss\_exponent \* np.log10(distances / reference\_distance)

# Define the path loss exponent symbol

path\_loss\_exponent = Symbol("alpha")

# Calculate path loss estimation

path\_loss\_estimate = calculate\_path\_loss\_estimate(distance\_values, reference\_path\_loss, reference\_distance, path\_loss\_exponent)

# Minimum mean square error

MSE = (np.mean(path\_loss\_matrix, axis=1) - path\_loss\_estimate) \*\* 2

dMSE = 0

# Derivative of MSE wrt alpha

for i in range(len(MSE)):

    dMSE += diff(MSE[i], path\_loss\_exponent )

alpha\_est = solve(dMSE, path\_loss\_exponent )

# Estimate the variance:

total\_samples=250

MSE\_alpha=13204.66516 # Substitude alpha into the MSE Equation

variance\_sq = MSE\_alpha/(total\_samples-1)

variance = math.sqrt(variance\_sq)

# Individual measurements

plt.scatter(np.tile(distance\_values, 50), path\_loss\_matrix, s=10, alpha=0.5, label="Individual Measurements")

# Mean path loss

mean\_path\_loss = np.mean(path\_loss\_matrix, axis=1)

plt.plot(distance\_values, mean\_path\_loss, marker="o", markersize=10, color="red", label="Mean Path Loss")

# Estimated path loss model

d = np.linspace(min(distance\_values), max(distance\_values), 100)

best\_fit = calculate\_path\_loss\_estimate(d, reference\_path\_loss, reference\_distance, alpha\_est[0])

plt.plot(d, best\_fit, linestyle="--", color="green", label="Estimated Path Loss Model")

plt.title("Path Loss Model", fontsize=16)

plt.xlabel("Distance [meters]", fontsize=12)

plt.ylabel("Path Loss [dB]", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

plt.legend(fontsize=12)

plt.tight\_layout()

plt.show()

print("Theoretical Value of Alpha: 4")

print("Estimated Value of Alpha: ", alpha\_est[0])

print("Theoretical Value of Variance: 8")

print("Estimated Value of Alpha: ", variance)

### Output

A graph with red and green lines

Description automatically generated

Derivative of MSE:

5897.20480686091*α*−23573.2427533061

Estimated Values:

**Theoretical Value of Alpha: 4**

**Estimated Value of Alpha: 3.99735866827630**

**Theoretical Value of Variance: 8**

**Estimated Value of Alpha: 7.282223820722651**

# Question 7

## part a

Generate 200 samples of a Rayleigh random variable R with E{R2}=1. Plot the samples in a stem plot. Remember that R = |X+jY|, where X and Y are zero mean, independent Gaussian random variables (r.v). Your Gaussian r.v. X and Y (produced by randn command) must each have equal variance equal to ½.

num\_samples = 200

mean = 0

variance = 1/2

X = np.random.normal(mean, np.sqrt(variance), num\_samples)

Y = np.random.normal(mean, np.sqrt(variance), num\_samples)

R = np.abs(X + 1j \* Y)

plt.figure(figsize=(8, 6))

plt.stem(range(num\_samples), R, basefmt=" ", linefmt="-", markerfmt="black")

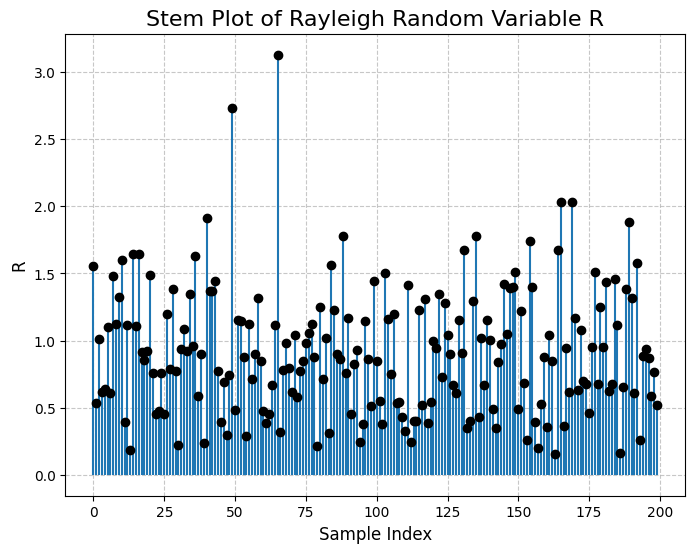
plt.xlabel('Sample Index', fontsize=12)

plt.ylabel('R', fontsize=12)

plt.title('Stem Plot of Rayleigh Random Variable R', fontsize=16)

plt.grid(True, linestyle='--', alpha=0.7)

plt.show()



## Part b

Give the estimated rms value of R, based on the 200 samples generated, which is given as .

# Calculate R as the absolute value of (X + jY)

R = np.abs(X + 1j \* Y)

# Calculate the estimated RMS value of R

estimated\_rms = np.sqrt(np.mean(R\*\*2))

print(estimated\_rms)

1.0332093785368484

## part C

What fraction (if any) of these Rayleigh samples are 10dB below the estimated rms value?(Note that this threshold corresponds to in the context of level crossing)

threshold\_db = -10  # 10 dB below

threshold\_value = estimated\_rms \* 10\*\*(threshold\_db / 10)

below\_threshold\_samples = np.sum(R < threshold\_value)

fraction\_below\_threshold = below\_threshold\_samples / num\_samples

print("Estimated RMS value of R:", estimated\_rms)

print("Threshold Value:", threshold\_value)

print("Fraction of samples below the threshold:", fraction\_below\_threshold)

Estimated RMS value of R: 1.0332093785368484

Threshold Value: 0.32672949360235304

Fraction of samples below the threshold: 0.08

## part d

Generate 200 samples of a Rician random variable by adding means mr=5cos(π/3) and mi = 5sin(π/3), respectively to the real part (X) and imaginary part (Y) in part (a). Plot the samples in stem plot. What is the K factor of this Rician random variable?

# Number of samples

num\_samples = 200

# Define parameters for the Rician distribution

mean\_real = 5 \* math.cos(math.pi / 3)

mean\_imag = 5 \* math.sin(math.pi / 3)

variance = 0.5

rand\_rician = (mean\_real + np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples)) + (

    1j \* (np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples) + mean\_imag)

)

samples\_magnitudes = np.abs(rand\_rician)

plt.figure(figsize=(10, 6))

plt.stem(range(num\_samples), samples\_magnitudes, basefmt=" ", linefmt="-", markerfmt="b.")

plt.xlabel('Sample Index', fontsize=12)

plt.ylabel('Magnitude', fontsize=12)

plt.title('Stem Plot of Rician Distribution Samples', fontsize=16)

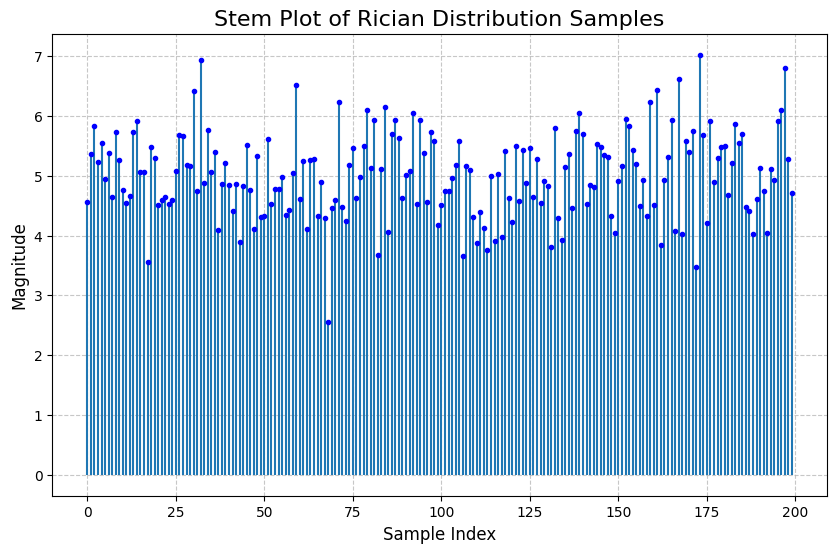
plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

# Calculate the K factor

K = np.abs(mean\_real + 1j \* mean\_imag) \*\* 2 / (2 \* np.var(samples\_magnitudes))

print("K Factor:", K)



K Factor: 25.027463862685803

## part e

Repeat part (d) except use mr=5cos(π/6) and mi = 5sin(π/6). Plot the samples in stem plot. What is the effect of phase change on the appearance of stem plot?

# Number of samples

num\_samples = 200

# Define parameters for the Rician distribution

mean\_real = 5 \* math.cos(math.pi / 6)

mean\_imag = 5 \* math.sin(math.pi / 6)

variance = 0.5

rand\_rician = (mean\_real + np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples)) + (

    1j \* (np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples) + mean\_imag)

)

samples\_magnitudes = np.abs(rand\_rician)

plt.figure(figsize=(10, 6))

plt.stem(range(num\_samples), samples\_magnitudes, basefmt=" ", linefmt="-", markerfmt="b.")

plt.xlabel('Sample Index', fontsize=12)

plt.ylabel('Magnitude', fontsize=12)

plt.title('Stem Plot of Rician Distribution Samples', fontsize=16)

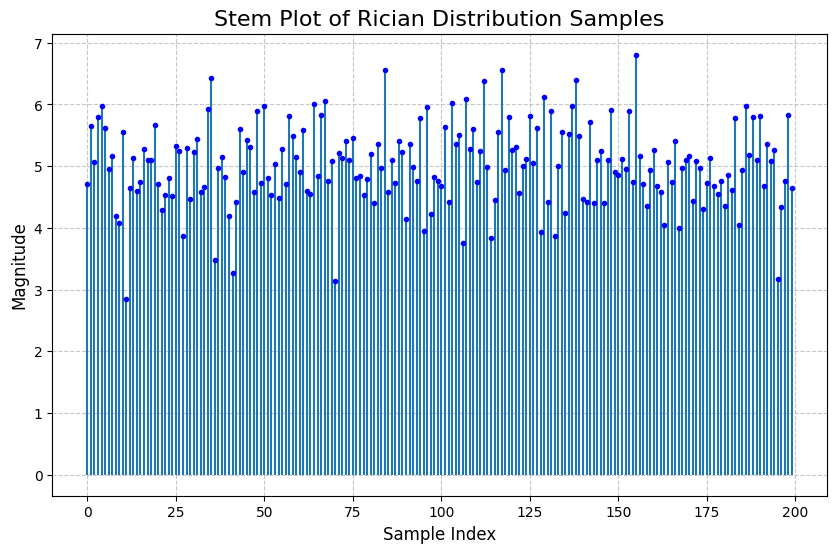
plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

# Calculate the K factor

K = np.abs(mean\_real + 1j \* mean\_imag) \*\* 2 / (2 \* np.var(samples\_magnitudes))

print("K Factor:", K)



K Factor: 24.58917390905922

## part f

We now normalize the Rician random variables to have unit mean square value. Let Rn be the nth sample from Part (d). Make 200 normalized r.v. as . Plot the Wn’s as a stem plot and compare to Part (a). What fraction of samples of W are 10dB below the rms value of W (should be fewer, because there should be less fading).

num\_samples = 200

mean\_real = 5 \* math.cos(math.pi / 3)

mean\_imag = 5 \* math.sin(math.pi / 3)

variance = 0.5

rand\_rician = (mean\_real + np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples)) + (

    1j \* (np.random.normal(loc=0, scale=np.sqrt(variance), size=num\_samples) + mean\_imag)

)

samples\_magnitudes = np.abs(rand\_rician)

normalized\_samples = rand\_rician / np.sqrt(np.square(np.abs(rand\_rician)).mean())

# Create a stem plot for the normalized samples

plt.figure(figsize=(10, 6))

plt.stem(range(num\_samples), np.abs(normalized\_samples), basefmt=" ", linefmt="-", markerfmt="r.")

plt.xlabel('Sample Index', fontsize=12)

plt.ylabel('Normalized Magnitude', fontsize=12)

plt.title('Stem Plot of Normalized Rician Samples', fontsize=16)

plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

rms\_value\_normalized = np.sqrt(np.square(np.abs(normalized\_samples)).mean())

threshold = 10  # dB below RMS

threshold\_value = 10 \*\* (threshold / 20) \* rms\_value\_normalized

fraction\_below\_threshold = np.sum(np.abs(normalized\_samples) > threshold\_value) / num\_samples

print(f"Fraction of samples 10dB below the RMS value: {fraction\_below\_threshold:.2%}")

print("RMS value is ", rms\_value\_normalized)

A diagram of a stem plot

Description automatically generated

Fraction of samples 10dB below the RMS value: 0.00%

RMS value is 1.0