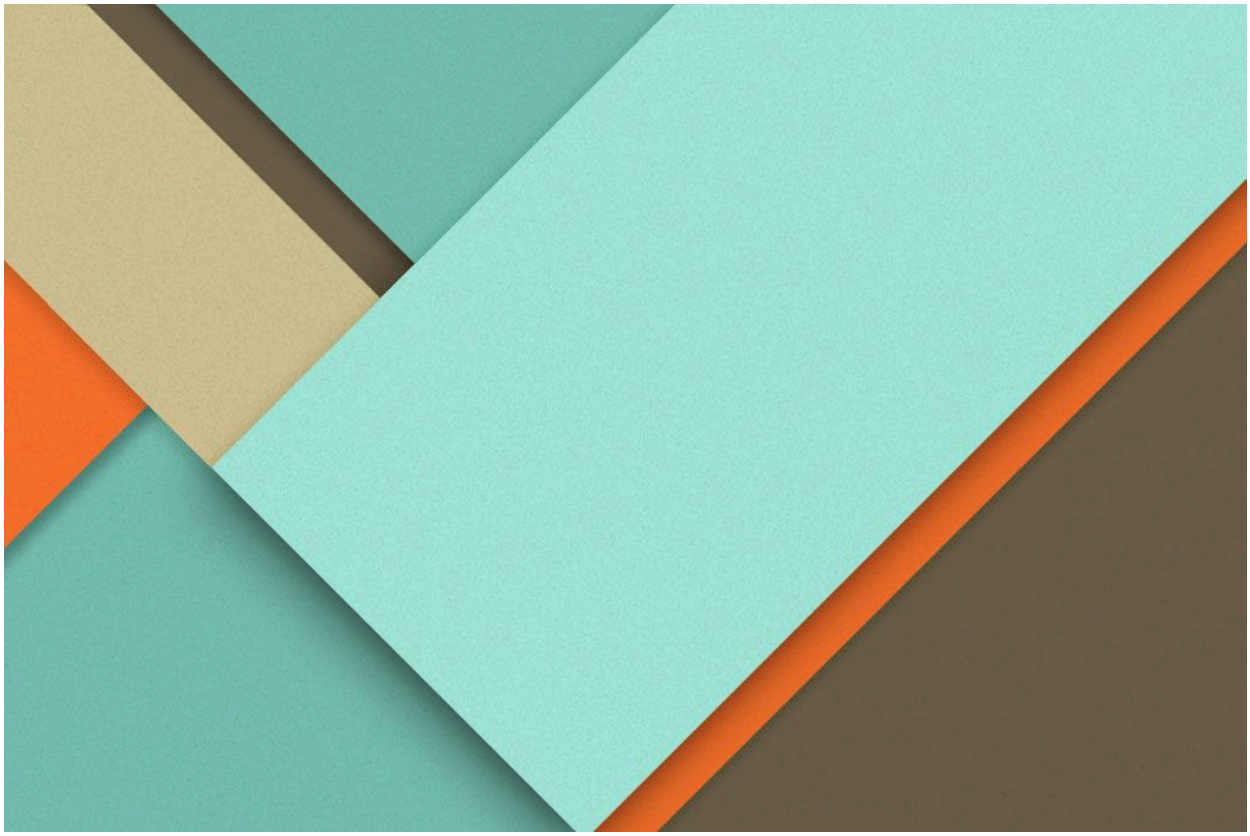


Detection & Estimation Theory: Assignment-4

25.11.2025

—

Ishan Jha
IMT2022562
International Institute of Information Technology Bangalore (IIIT-B)

GitHub Link :

If, for any reason, any of the codes are not downloadable or if there is an issue downloading the code, please clink on the following GitHub repository link, where you can find all the codes (as well as the report) which can be easily downloaded and executed on your local system :

https://github.com/Ishaniiitb/Detection_Estimation_Theory_Assignment_4

Problem-1

The first problem statement of the assignment was as follows :

1. Write a code to estimate DC level from N number of IID zero mean unit variance Gaussian noisy samples (the classical example discussed in class)
    a. Choose a wide range of values for N (say 10, 100, 1000 so on) and show the effect of number of samples on the efficacy of the estimator. Hint: Plot the CRLB against the empirically evaluated MSE for each case
    b. Now say the noise samples are non-IID zero mean Gaussian with a covariance matrix \Sigma. How does a sample mean estimator perform? Will it still be MVUE? Why? Justify both theoretically and through simulation.
    c. For IID Gaussian noise samples case, choose any other estimator other than the optimal sample mean estimator. Demonstrate via simulation that it is not MVUE

Part-A

For this part of the question, we proceed with the assumption that:

$$y[n] = A + w[n] \; ; \; n=\{0,...,N-1\}$$

with

$$w[n] \sim N(0 \; , \; \sigma^2) \text{ i.i.d (Identical \& Independently Distributed)}$$

Thus giving us the expression :

$$y[n] \sim N(A \; , \; \sigma^2)$$

So the input program to run a simulation of this part is given as follows :

```
# Name - Ishan Jha
```

```python
# Roll - IMT2022562

# Date - 25/11/2025

# Question - 1(a)


import matplotlib.pyplot as plt

import numpy as np

import random


def display(true, sigma, trials):

    print("Following are the quantities for the current simulation : ")

    print("1) True Value        : ", true)

    print("2) Variance of Noise : ", sigma)

    print("3) Number of Trials  : ", trials)


print("================================================================")


def compute_crlb(noise_power, sample_count):

    return noise_power / sample_count                                  #
Theoretical CRLB = σ²/N.


def generate_dc_measurements(true_bias, noise_dev, total_samples):

    noise_vec = np.random.normal(0, noise_dev, total_samples)

    return true_bias + noise_vec                                       #
Generate x[n] = A + w[n]


def estimate_dc_level(data_vector):

    """Use the sample average as estimator."""

    return np.mean(data_vector)
```

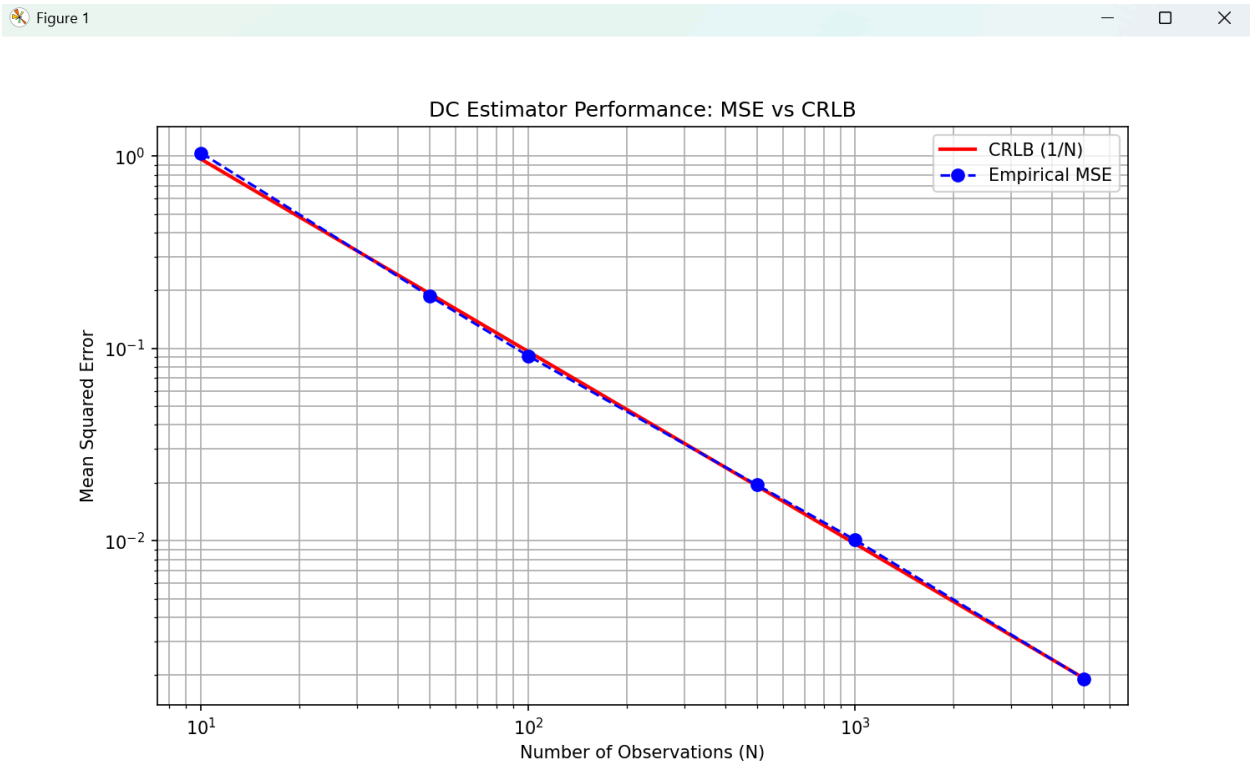
```
def run_dc_trials(actual_offset, noise_dev, sample_count, trial_count):

    estimates = []                                                  # Run
multiple iterations for DC estimation.

    for _ in range(trial_count):
        samples = generate_dc_measurements(actual_offset, noise_dev,
sample_count)
        est = estimate_dc_level(samples)
        estimates.append(est)

    estimates = np.array(estimates)
    mse = np.mean((estimates - actual_offset) ** 2)
    crlb_val = compute_crlb(noise_dev**2, sample_count)

    return mse, crlb_val, estimates

def plot_mse_vs_crlb(sample_sizes, mse_vals, crlb_vals):
    plt.figure(figsize=(10, 6))
    plt.loglog(sample_sizes, crlb_vals, 'r-', linewidth=2, label="CRLB
(1/N)")
    plt.loglog(sample_sizes, mse_vals, 'bo--', markersize=7,
label="Empirical MSE")
    plt.xlabel("Number of Observations (N)")
    plt.ylabel("Mean Squared Error")
    plt.title("DC Estimator Performance: MSE vs CRLB")
    plt.grid(True, which="both")
```

```
    plt.legend()
    plt.show()

def execute_dc_demo():

    true_value = random.uniform(2.5, 7.5)
    noise_sigma = random.uniform(0.5, 5)
    sample_grid = [10, 50, 100, 500, 1000, 5000]
    trials = 1000

    display(true_value, noise_sigma, trials)

    mse_results = []
    crlb_results = []

    print(f"{'N':<10} | {'MSE Estimate':<20} | {'CRLB':<20}")

print("============================================================")

    for N in sample_grid:
        mse, crlb, _ = run_dc_trials(true_value, noise_sigma, N, trials)
        mse_results.append(mse)
        crlb_results.append(crlb)

        print(f"{N:<10} | {mse:<20.6f} | {crlb:<20.6f}")

print("============================================================")
```

```
    plot_mse_vs_crlb(sample_grid, mse_results, crlb_results)



execute_dc_demo()
```

And the outputs obtained corresponding to the above code as as follows :

```
PS C:\Users\ishan\Desktop\IIITB\Detection & Estimation Theory\Assignment\Assignment_4\DET_Assignment_4\Ishan> & C:/Users/ishan/AppData/Local/Programs/Python/
Python313/python.exe "c:/Users/ishan/Desktop/IIITB/Detection & Estimation Theory/Assignment/Assignment_4/DET_Assignment_4/Ishan/Q1/Q1_a.py"
Following are the quantities for the current simulation :
1) True Value        :  3.0151778177220177
2) Variance of Noise :  3.1043879447369154
3) Number of Trials  :  1000
============================================================
N          | MSE Estimate         | CRLB
============================================================
10         | 1.038486             | 0.963722
50         | 0.186717             | 0.192744
100        | 0.091506             | 0.096372
500        | 0.019532             | 0.019274
1000       | 0.010083             | 0.009637
5000       | 0.001919             | 0.001927
============================================================
```



Part-B

The theoretical solution is as follows (the following solution was typed in LaTex and screenshotted and pasted here as it was difficult to insert all the special symbols involved in the solution directly on Word/Docs. The link to the LaTex solution is provided in the GitHub Repository) :

Model Description

Consider a sequence of N observations given by

$$y[n] = \mu + w[n], \qquad n = 1, \ldots, N,$$

where the noise samples are collected in the vector

$$\mathbf{w} = [w[1],\, w[2],\, \ldots,\, w[N]]^T.$$

Unlike the i.i.d. case, we assume these noise terms follow a multivariate Gaussian distribution

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma),$$

with Σ being a known, symmetric, positive-definite covariance matrix. Its entries satisfy

$$\Sigma[n, m] = \mathrm{Cov}(w[n], w[m]) = \mathrm{Cov}(y[n], y[m]).$$

Letting

$$\mathbf{y} = [y[1], \ldots, y[N]]^T, \qquad \boldsymbol{\mu} = [\mu, \ldots, \mu]^T,$$

the observation vector obeys

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma).$$

Sample Mean: Properties

The standard arithmetic mean is

$$\hat{\mu}_{\mathrm{SM}} = \frac{1}{N} \sum_{n=1}^{N} y[n].$$

Expectation Using linearity,

$$\mathbb{E}[\hat{\mu}_{\mathrm{SM}}] = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[y[n]] = \frac{1}{N} \sum_{n=1}^{N} \mu = \mu,$$

so the sample mean remains unbiased regardless of the dependency structure in Σ.

Variance The variance can be expanded as

$$\begin{aligned}\mathrm{Var}(\hat{\mu}_{\mathrm{SM}}) &= \mathrm{Var}\left(\frac{1}{N}\sum_{n=1}^{N} y[n]\right) \\ &= \frac{1}{N^2}\,\mathrm{Var}\left(\sum_{n=1}^{N} y[n]\right) \\ &= \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\mathrm{Cov}(y[n], y[m]) \\ &= \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\Sigma[n,m].\end{aligned}$$

Thus,

$$\boxed{\mathrm{Var}(\hat{\mu}_{\mathrm{SM}}) = \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\Sigma[n,m]}.$$

Likelihood Function and Fisher Information

The probability density of $\mathbf{y}$ under the model is

$$f(\mathbf{y};\mu) = \frac{\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^N \det(\Sigma)}}.$$

Taking the natural logarithm yields

$$\log f(\mathbf{y};\mu) = -\frac{1}{2}\ln\left((2\pi)^N \det\Sigma\right) - \frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{y}-\boldsymbol{\mu}).$$

Since

$$\frac{d\boldsymbol{\mu}}{d\mu} = [1,\ldots,1]^T,$$

the derivative of the log-likelihood becomes

$$\begin{aligned}\frac{d}{d\mu}\log f(\mathbf{y};\mu) &= (\mathbf{y}-\boldsymbol{\mu})^T\Sigma^{-1}\frac{d\boldsymbol{\mu}}{d\mu} \\ &= \sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]\,(y[m]-\mu).\end{aligned}$$

Define the scalar quantity

$$S \triangleq \sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m].$$

Fisher Information Using $\mathbb{E}[(y[m]-\mu)(y[k]-\mu)] = \Sigma[m,k]$,

$$I(\mu) = \mathbb{E}\left[\left(\frac{d}{d\mu}\log f(\mathbf{y};\mu)\right)^2\right] = \sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m].$$

Thus,

$$\boxed{I(\mu) = \sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]}.$$

CRLB For any unbiased estimator $\hat{\mu}$,

$$\boxed{\mathrm{Var}(\hat{\mu}) \geq \frac{1}{I(\mu)} = \left(\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]\right)^{-1}}.$$

The GLS (MLE) Estimator

The optimal estimator is obtained from the condition

$$\frac{d}{d\mu}\log f(\mathbf{y};\mu) = 0,$$

leading to

$$\sum_{m=1}^{N}\left(\sum_{n=1}^{N}(\Sigma^{-1})[n,m]\right)y[m] = \mu\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m].$$

Hence the estimator is

$$\boxed{\hat{\mu}_{\mathrm{GLS}} = \frac{\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]\,y[m]}{\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]}}.$$

This estimator satisfies

$$\mathbb{E}[\hat{\mu}_{\mathrm{GLS}}] = \mu$$

and its variance is

$$\mathrm{Var}(\hat{\mu}_{\mathrm{GLS}}) = \frac{1}{I(\mu)}.$$

Thus it attains the CRLB and is an efficient (and MVUE) estimator.

Is the Sample Mean MVUE?

Recall:

$$\mathrm{Var}(\hat{\mu}_{\mathrm{SM}}) = \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\Sigma[n,m], \qquad \mathrm{CRLB} = \left(\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]\right)^{-1}.$$

A known inequality involving Σ and Σ^{-1} (e.g., via eigen-decomposition or Cauchy–Schwarz) gives

$$\left(\sum_{n=1}^{N}\sum_{m=1}^{N}\Sigma[n,m]\right)\left(\sum_{n=1}^{N}\sum_{m=1}^{N}(\Sigma^{-1})[n,m]\right) \geq N^2,$$

which implies

$$\mathrm{Var}(\hat{\mu}_{\mathrm{SM}}) \geq \mathrm{CRLB}.$$

Thus,

$\hat{\mu}_{\mathrm{SM}}$ does not generally achieve the CRLB and is not the MVUE.

So the input program to run a simulation of this part is given as follows :

```
# Name - Ishan Jha
# Roll - IMT2022562
# Date - 25/11/2025
# Question - 1(b)


import numpy as np
import random
import matplotlib.pyplot as plt


def display(mu, variance, rho, samples, trials):
    print("Following are the quantities for the current simulation : ")
    print("1) True Value                 : ", mu)
    print("2) Variance                   : ", variance)
    print("3) Rho                        : ", rho)
    print("4) No of samples              : ", samples)
    print("5) No of trials/iterations    : ", trials)
```

```
print("================================================================")

def generate_toeplitz_cov(size, rho, variance):
    idx = np.arange(size)
    return variance * (rho ** np.abs(idx[:, None] - idx[None, :]))

def gls_estimator(y_vec, cov_inv):
    ones = np.ones(len(y_vec))
    denominator = ones @ cov_inv @ ones
    return (ones @ cov_inv @ y_vec) / denominator

def run_correlated_experiment(true_mu, variance, rho, N, trials):
    Sigma = generate_toeplitz_cov(N, rho, variance)
    Sigma_inv = np.linalg.inv(Sigma)

    ones = np.ones(N)
    denom = ones @ Sigma_inv @ ones
    crlb_val = 1.0 / denom

    sample_mean_vals = []
    gls_vals = []

    L = np.linalg.cholesky(Sigma)

    for _ in range(trials):
        z = np.random.randn(N)
        noise = L @ z
```

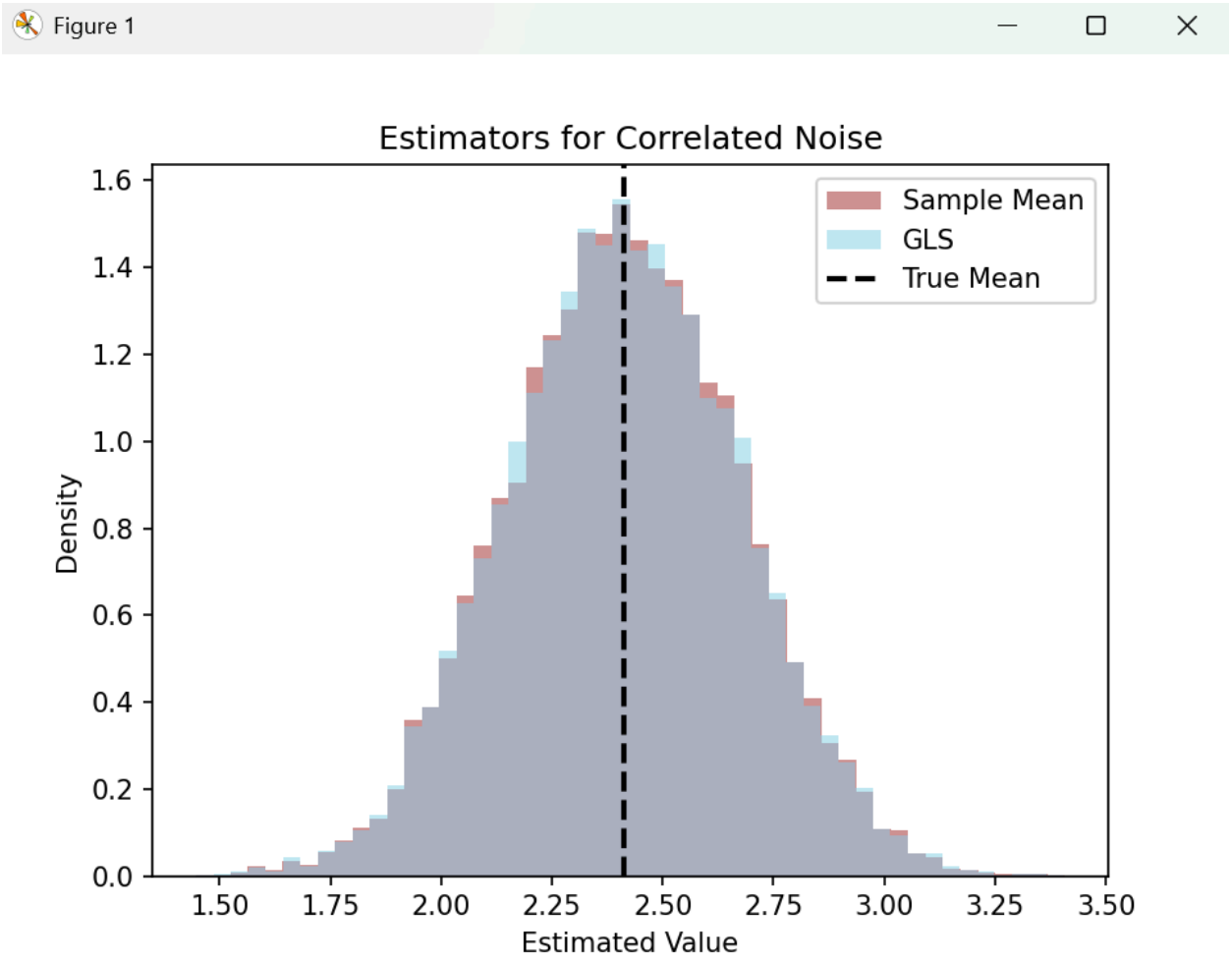
```
        obs = true_mu + noise


        sample_mean_vals.append(np.mean(obs))
        gls_vals.append(gls_estimator(obs, Sigma_inv))


    sm = np.array(sample_mean_vals)
    gls = np.array(gls_vals)


    return sm, gls, crlb_val


def compare_histograms(sm_arr, gls_arr, true_mu):
    plt.figure()
    plt.hist(sm_arr, bins=50, alpha=0.5, density=True, label="Sample
Mean", color="brown")
    plt.hist(gls_arr, bins=50, alpha=0.5, density=True, label="GLS",
color="skyblue")
    plt.axvline(true_mu, color='k', linestyle='--', linewidth=2,
label="True Mean")
    plt.title("Estimators for Correlated Noise")
    plt.xlabel("Estimated Value")
    plt.ylabel("Density")
    plt.legend()
    plt.show()


def correlated_noise_simulation():


    true_mu = random.uniform(1, 5)
    variance = random.uniform(0.5, 5)
```

```
    rho = random.uniform(0.1, 1)
    N = 100
    trials = 10000

    display(true_mu, variance, rho, N, trials)

    sm_arr, gls_arr, crlb_val = run_correlated_experiment(true_mu,
variance, rho, N, trials)

    mse_sm = np.mean((sm_arr - true_mu) ** 2)
    mse_gls = np.mean((gls_arr - true_mu) ** 2)

    print(f"MSE (Sample Mean) : {mse_sm:.6f}")
    print(f"MSE (GLS)         : {mse_gls:.6f}")
    print(f"CRLB              : {crlb_val:.6f}")

    compare_histograms(sm_arr, gls_arr, true_mu)


correlated_noise_simulation()
```

And the outputs obtained corresponding to the above code as as follows :

```
PS C:\Users\ishan\Desktop\IIITB\Detection & Estimation Theory\Assignment\Assignment_4\DET_Assignment_4\Ishan> & C:/Users/ishan/AppData/Local/Programs/Python/
Python313/python.exe "c:/Users/ishan/Desktop/IIITB/Detection & Estimation Theory/Assignment/Assignment_4/DET_Assignment_4/Ishan/Q1/Q1_b.py"
Following are the quantities for the current simulation :
1) True Value               :  2.4130170234507005
2) Variance                 :  4.20553018736119
3) Rho                      :  0.23891824230644282
4) No of samples            :  100
5) No of trials/iterations  :  10000
============================================================
MSE (Sample Mean) : 0.069118
MSE (GLS)         : 0.069010
CRLB              : 0.068032
```

As we can see in the above results, the variance obtained by using the sample mean estimator is greater than obtained using the CRLB method, thus proving to us through simulation that sample mean is not the MVUE in the case of correlated noise.



Part-C

The theoretical solution is as follows (the following solution was typed in LaTex and screenshotted and pasted here as it was difficult to insert all the special symbols involved in the solution directly on Word/Docs. The link to the LaTex solution is provided in the GitHub Repository) :

i.i.d. Gaussian Observation Model

We revisit the situation in which the measurements arise from the model

$$y[n] = \mu + w[n], \qquad n = 1, \ldots, N,$$

where the noise samples $w[n]$ are independent and identically distributed according to

$$w[n] \sim \mathcal{N}(0, \sigma^2).$$

Thus each observation $y[n]$ is also Gaussian:

$$y[n] \sim \mathcal{N}(\mu, \sigma^2).$$

Since the samples are i.i.d., the Fisher information regarding the parameter μ is well known:

$$I(\mu) = \frac{N}{\sigma^2}.$$

The Cramér–Rao lower bound then guarantees that any unbiased estimator $\hat{\mu}$ must obey

$$\mathrm{Var}(\hat{\mu}) \geq \frac{\sigma^2}{N}.$$

Efficiency of the Sample Mean

Consider the sample-mean estimator

$$\hat{\mu}_{\mathrm{SM}} = \frac{1}{N} \sum_{n=1}^{N} y[n].$$

Its variance is easily computed:

$$\mathrm{Var}(\hat{\mu}_{\mathrm{SM}}) = \frac{\sigma^2}{N}.$$

Since this variance exactly matches the CRLB, the sample mean is not only unbiased but also efficient (and consequently the MVUE in this setting).

A Different Unbiased Estimator

To illustrate that not all unbiased estimators achieve the CRLB, let us examine a much simpler and much less effective estimator:

$$\hat{\mu}_{\text{alt}} = y[1].$$

Because each sample has mean μ, we immediately obtain

$$\mathbb{E}[\hat{\mu}_{\text{alt}}] = \mathbb{E}[y[1]] = \mu,$$

so $\hat{\mu}_{\text{alt}}$ is unbiased.

However, its variance is simply the variance of a single observation:

$$\text{Var}(\hat{\mu}_{\text{alt}}) = \text{Var}(y[1]) = \sigma^2.$$

For any $N > 1$,

$$\sigma^2 > \frac{\sigma^2}{N},$$

which demonstrates that

$$\text{Var}(\hat{\mu}_{\text{alt}}) > \text{Var}(\hat{\mu}_{\text{SM}}).$$

Thus the estimator $\hat{\mu}_{\text{alt}}$ is unbiased but decidedly non-optimal, as it fails to achieve the Cramér–Rao lower bound.

So the input program to run a simulation of this part is given as follows :

```
# Name - Ishan Jha

# Roll - IMT2022562

# Date - 25/11/2025

# Question - 1(c)


import matplotlib.pyplot as plt

import numpy as np

import random


def display(true, sigma, samples, trials):

    print("Following are the quantities for the current simulation : ")

    print("1) True Value                 : ", true)
```

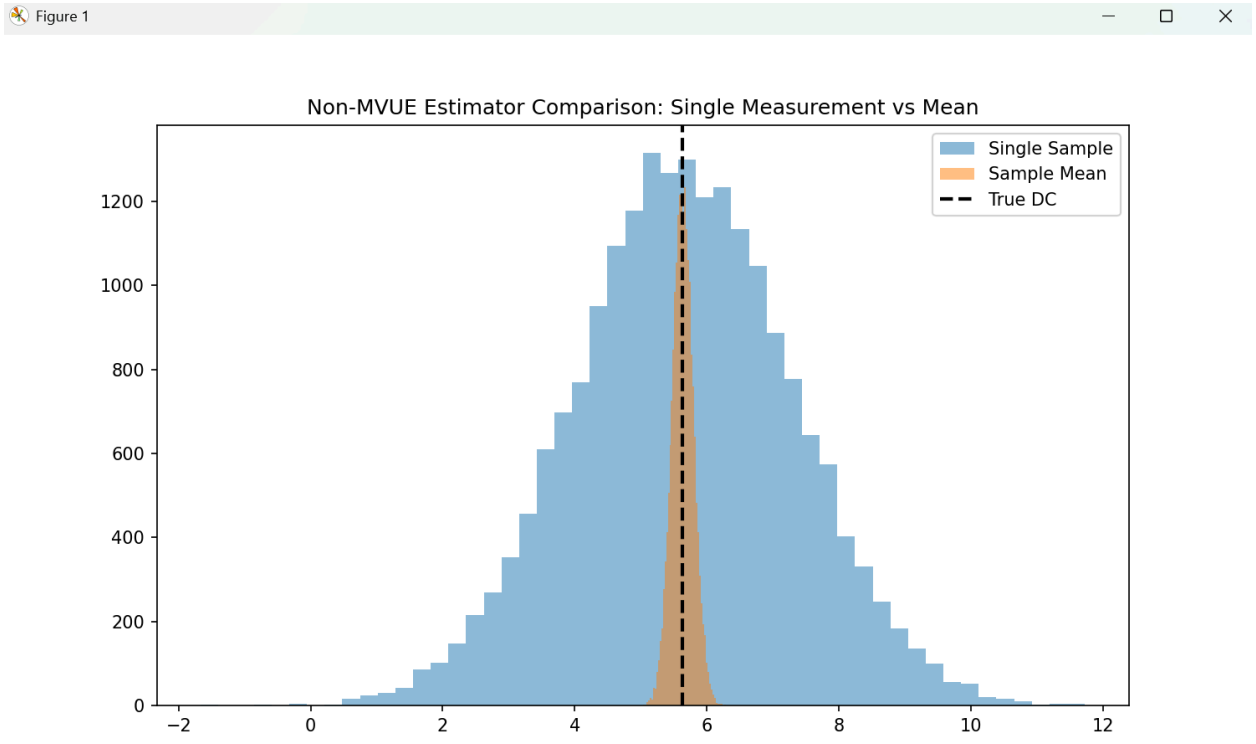
```
    print("2) Variance                  : ", sigma**2)
    print("3) No of samples             : ", samples)
    print("4) No of trials/iterations   : ", trials)

print("================================================================")

def generate_noisy_dc(A, noise_std, N):
    noise_vec = np.random.normal(0, noise_std, N)
    return A + noise_vec

def estimator_mean(data):
    return np.mean(data)

def estimator_single(data):
    return data[0]

def run_suboptimal_trials(true_val, sigma, N, runs):

    mean_list = []
    single_list = []

    for _ in range(runs):
        samples = generate_noisy_dc(true_val, sigma, N)
        mean_list.append(estimator_mean(samples))
        single_list.append(estimator_single(samples))

    return np.array(mean_list), np.array(single_list)
```

```
def plot_distribution_comparison(mean_arr, single_arr, true_val):
    plt.figure(figsize=(10, 6))
    plt.hist(single_arr, bins=50, alpha=0.5, label="Single Sample")
    plt.hist(mean_arr, bins=50, alpha=0.5, label="Sample Mean")
    plt.axvline(true_val, color='k', linestyle='--', linewidth=2,
label="True DC")
    plt.title("Non-MVUE Estimator Comparison: Single Measurement vs Mean")
    plt.legend()
    plt.show()


def nonMVUE_simulation():


    true_dc = random.uniform(2.5, 7.5)
    sigma = random.uniform(0.5, 2.25)
    N = 100
    runs = 20000


    display(true_dc, sigma, N, runs)


    mean_arr, single_arr = run_suboptimal_trials(true_dc, sigma, N, runs)


    mse_mean = np.var(mean_arr)
    mse_single = np.var(single_arr)


    print("Non-MVUE Estimator Simulation : ")
    print(f"MSE (Single Sample) : {mse_single:.4f}")
    print(f"MSE (Sample Mean)   : {mse_mean:.4f}")
    print(f"CRLB (Single)       : {sigma**2:.4f}")
```

```
    print(f"CRLB (Mean)          : {sigma**2 / N:.4f}")

    plot_distribution_comparison(mean_arr, single_arr, true_dc)



nonMVUE_simulation()
```

And the outputs obtained corresponding to the above code as as follows :

```
PS C:\Users\ishan\Desktop\IIITB\Detection & Estimation Theory\Assignment\Assignment_4\DET_Assignment_4\Ishan> & C:/Users/ishan/AppData/Local/Programs/Python/
Python313/python.exe "c:/Users/ishan/Desktop/IIITB/Detection & Estimation Theory/Assignment/Assignment_4/DET_Assignment_4/Ishan/Q1/Q1_c.py"
Following are the quantities for the current simulation :
1) True Value               :  5.6352178088563925
2) Variance                 :  2.7280266007417437
3) No of samples            :  100
4) No of trials/iterations  :  20000
============================================================
Non-MVUE Estimator Simulation :
MSE (Single Sample) : 2.6956
MSE (Sample Mean)   : 0.0270
CRLB (Single)       : 2.7280
CRLB (Mean)         : 0.0273
```



Problem-2

The second problem statement of the assignment was as follows :

Theorem 4.2 (Minimum Variance Unbiased Estimator for General Linear Model) *If the data can be modeled as*

$$\mathbf{x} = \mathbf{H}\boldsymbol{\theta} + \mathbf{s} + \mathbf{w} \tag{4.30}$$

where $\mathbf{x}$ *is an* $N \times 1$ *vector of observations,* $\mathbf{H}$ *is a known* $N \times p$ *observation matrix* $(N > p)$ *of rank* p, $\boldsymbol{\theta}$ *is a* $p \times 1$ *vector of parameters to be estimated,* $\mathbf{s}$ *is an* $N \times 1$ *vector of known signal samples, and* $\mathbf{w}$ *is an* $N \times 1$ *noise vector with PDF* $\mathcal{N}(\mathbf{0}, \mathbf{C})$, *then the MVU estimator is*

$$\hat{\boldsymbol{\theta}} = (\mathbf{H}^T\mathbf{C}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{C}^{-1}(\mathbf{x} - \mathbf{s}) \tag{4.31}$$

and the covariance matrix is

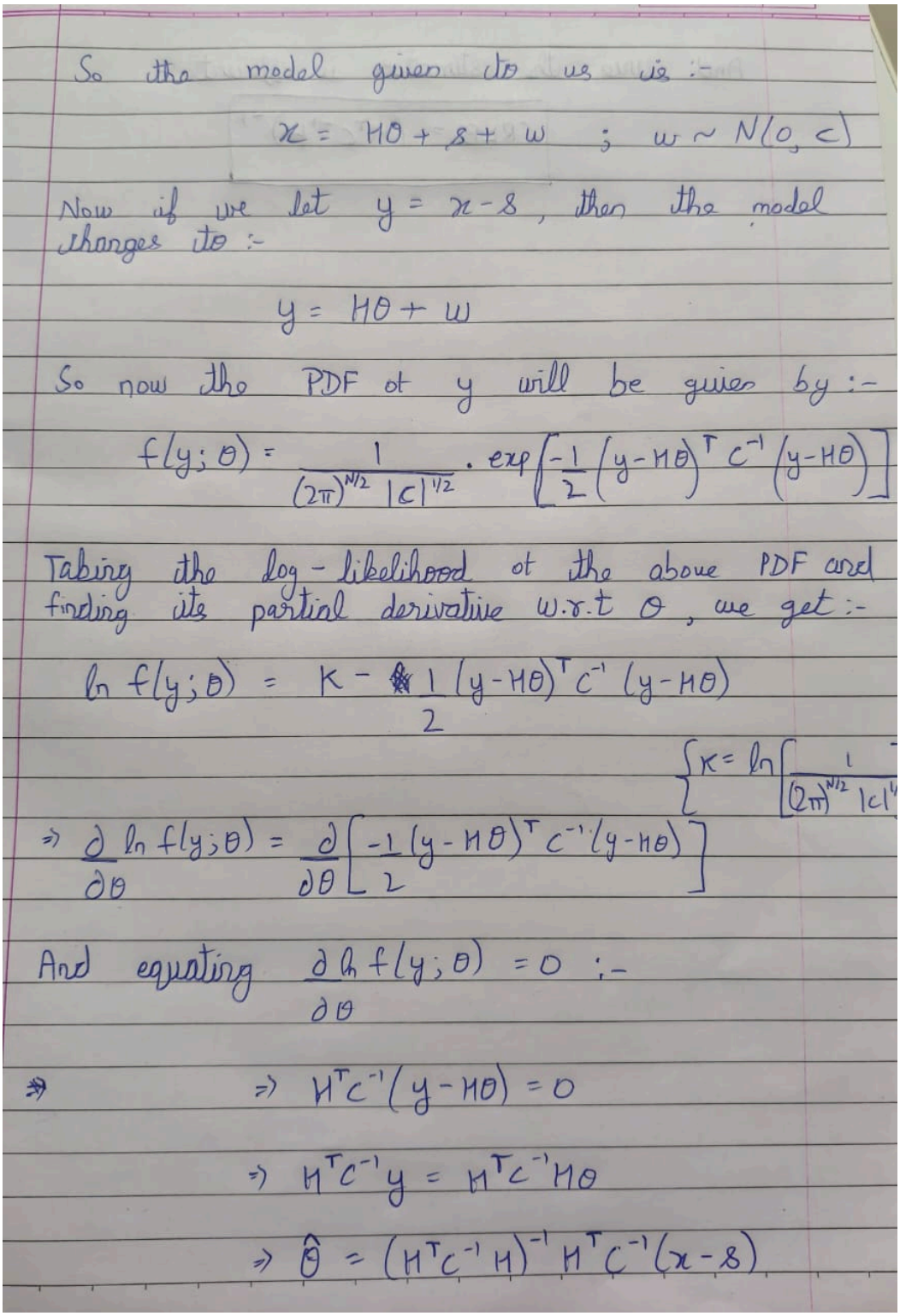
$$\mathbf{C}_{\hat{\theta}} = (\mathbf{H}^T\mathbf{C}^{-1}\mathbf{H})^{-1}. \tag{4.32}$$

For the general linear model the MVU estimator is efficient in that it attains the CRLB.

a. Derive the expressions of the estimator, covariance matrix and CRLB.
b. Demonstrate via simulations that the above estimator is indeed an efficient estimator.

Part-A

For this part, the theoretical solution is given as follows :



So the model given to us is :-

$$x = H\theta + s + w \quad ; \quad w \sim N(0, C)$$

Now if we let $y = x - s$, then the model changes to :-

$$y = H\theta + w$$

So now the PDF of y will be given by :-

$$f(y;\theta) = \frac{1}{(2\pi)^{N/2}|C|^{1/2}} \cdot \exp\left[\frac{-1}{2}(y-H\theta)^T C^{-1}(y-H\theta)\right]$$

Taking the log-likelihood of the above PDF and finding its partial derivative w.r.t θ, we get :-

$$\ln f(y;\theta) = K - \frac{1}{2}(y-H\theta)^T C^{-1}(y-H\theta)$$

$$\left\{K = \ln\left[\frac{1}{(2\pi)^{N/2}|C|^{1/2}}\right]\right.$$

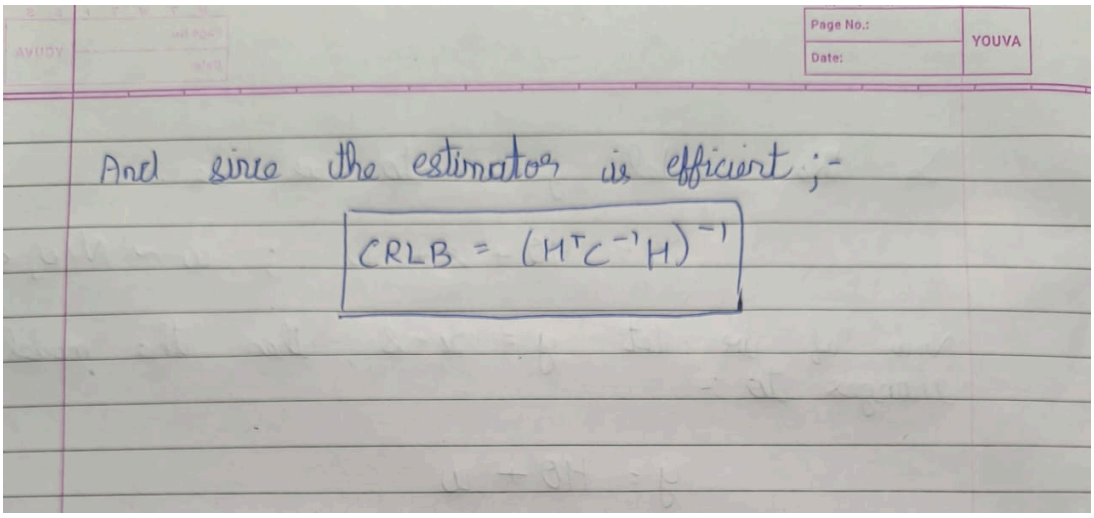
$$\Rightarrow \frac{\partial \ln f(y;\theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left[\frac{-1}{2}(y-H\theta)^T C^{-1}(y-H\theta)\right]$$

And equating $\frac{\partial \ln f(y;\theta)}{\partial \theta} = 0$:-

$$\Rightarrow H^T C^{-1}(y-H\theta) = 0$$

$$\Rightarrow H^T C^{-1} y = H^T C^{-1} H\theta$$

$$\Rightarrow \hat{\theta} = (H^T C^{-1} H)^{-1} H^T C^{-1}(x-s)$$



Part-B

So the objective of the following program is to run a simulation of the estimator obtained in the above sub-question and verify if it is an efficient estimator :

```
# Name - Ishan Jha
# Roll - IMT2022562
# Date - 25/11/2025
# Question - 2



import numpy as np
import random
import matplotlib.pyplot as plt



sample_count = 100       # No of measurements
param_count = 2          # No of unknown parameters
trial_count = 20000      # No of iterations
```

```
def graphical_plots(estimates, thetas, crlb_dig):
    plt.figure(figsize=(10, 6))

    # Plot histogram
    _, bins, _ = plt.hist(estimates[:, 0],
                          bins=50, density=True,
                          alpha=0.6, color='green',
                          label='Simulated Estimatations')

    # Overlay theoretical Gaussian curve predicted by CRLB
    mean_val = thetas[0]
    std_dev = np.sqrt(crlb_dig[0])

    pdf_vals = (1 / (std_dev * np.sqrt(2 * np.pi))) * \
               np.exp(- (bins - mean_val)**2 / (2 * std_dev**2))

    plt.plot(bins, pdf_vals, linewidth=3,
             color='red', label='CRLB Predicted PDF')

    plt.axvline(mean_val, color='black',
                linestyle='--', label='True θ₀')

    plt.title(f'Demonstration of Estimator Efficiency for θ₀\n'
              f'(No of Samples={sample_count} & No of
Trials={trial_count})')
    plt.xlabel('Parameter Value')
    plt.ylabel('Density')
```

```
    plt.grid(alpha=0.5)
    plt.legend()
    plt.show()


def statistics(estimations, crlb_dig):
    cov = np.cov(estimations, rowvar=False)
    var = np.diag(cov)


    print ("Estimated Variance : ", var)

print("=================================================================
======")
    print("Variance Ratio (Empirical/CRLB) : ", (var/crlb_dig))

print("=================================================================
======")


def crlb_calculations(H, noise):
    noise_cov = noise @ noise.T
    noise_cov_inv = np.linalg.inv(noise_cov)
    crlb = np.linalg.inv(H.T @ noise_cov_inv @ H)                    #
Theoretical CRLB = (H^T C^-1 H)^-1
    crlb_diagonal = np.diag(crlb)


    print("CRLB (variance limits for the parameters) : ", crlb_diagonal)

print("=================================================================
======")
    return crlb, crlb_diagonal, noise_cov, noise_cov_inv
```

```
def estimations_func(H, signal, thetas, noise_cov, noise_cov_inv, crlb):
    estimations = []
    estimation_matrix = crlb @ H.T @ noise_cov_inv

    for i in range(trial_count):
        noise = np.random.multivariate_normal(np.zeros(sample_count),
noise_cov)
        observation = H @ thetas + signal + noise
        estimations.append(estimation_matrix @ (observation - signal))

    return np.array(estimations)

def q2_simulation():

    reals = []
    for i in range(param_count):
        reals.append(random.uniform(-5, 5))

    theta_real = np.array(reals)                          # Ground Truth
for 3 parameters

    np.random.seed(42)
    design_matrix = np.random.randn(sample_count, param_count)       #
Observation Matrix - H
    signal_vec = np.random.randn(sample_count)                       #
Deterministic signal component
    random_block = np.random.randn(sample_count, sample_count)       #
Generation of noise covariance matrix
```

```
    crlb, crlb_diagonal, noise_cov, noise_cov_inv =
crlb_calculations(design_matrix, random_block)
    parameter_estimates = estimations_func(design_matrix, signal_vec,
theta_real, noise_cov, noise_cov_inv, crlb)
    statistics(parameter_estimates, crlb_diagonal)
    graphical_plots(parameter_estimates, theta_real, crlb_diagonal)


q2_simulation()
```

So if we set the number of parameters to 2 (each signal has a 100 number of samples, and we run 20000 Monte-Carlo iterations to get our desired output), we get the following results :

```
PS C:\Users\ishan\Desktop\IIITB\Detection & Estimation Theory\Assignment\Assignment_4\DET_Assignment_4\Ishan> & C:/Users/ishan/AppData/Local/Programs/Python/
Python313/python.exe "c:/Users/ishan/Desktop/IIITB/Detection & Estimation Theory/Assignment/Assignment_4/DET_Assignment_4/Ishan/Q2/Q2.py"
CRLB (variance limits for the parameters) :  [0.02382742 0.00313744]
======================================================================
Estimated Variance :  [0.02418275 0.00315147]
======================================================================
Variance Ratio (Empirical/CRLB) :  [1.01491284 1.00446953]
======================================================================
```