Assignment 1

Estimation Theory

Irsh Vijay
21EC39055

1 Problem Statement

- Line Fitting
- Polynomial Fitting
- Comparing Mean and Max Estimators in DC Estimation

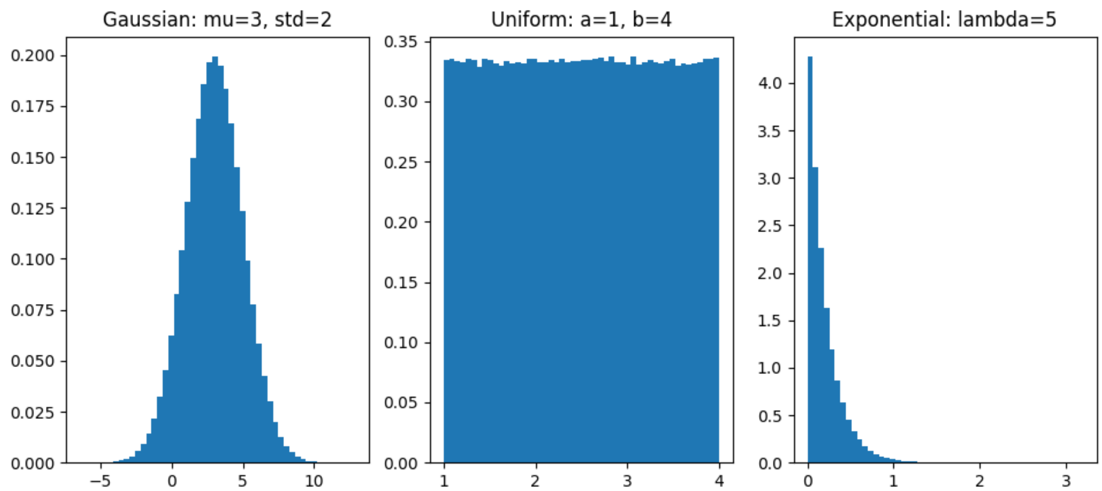
2 Code

2.1 Helper Functions

Random Variables:

```
class RandomVariable:
    def __init__(self) -> None:
        pass

    def uniform(self, a=0, b=1):
        assert b > a, "Upper Bound can't be lower!"
        return random.uniform(a, b)

    def exponential(self, l=2):
        # Monte Carlo Inversion
        return - np.log(self.uniform()) / l

    def gaussian(self, mu=0, sigma=1):
        # Box Muller
        u = self.uniform()
        v = self.uniform()
        r = np.sqrt(-2 * np.log(u))
        theta = 2 * np.pi * v
        X = r * np.cos(theta)
        return sigma*X + mu

    def distribution(self, samples, bins=10):
        plt.hist(samples, bins=bins, density=True)
        plt.title("Distribution Function")

    def show_plot(self):
        plt.show()
```

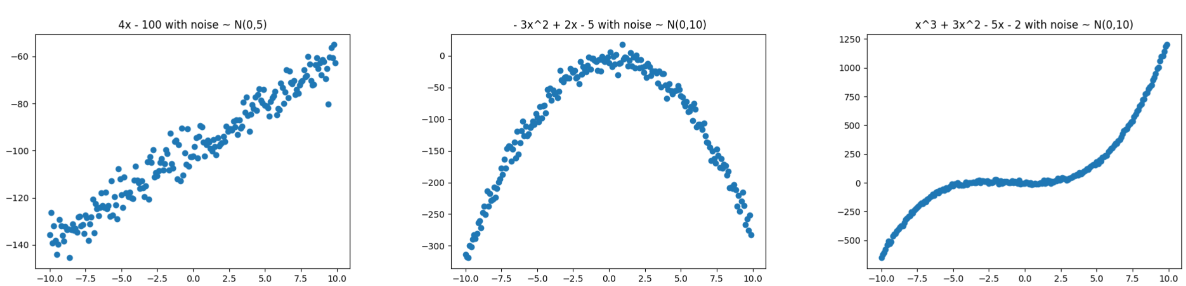
Distributions were generated using Monte Carlo Inversion / Box Muller Transform from a uniform random variable.



Polynomials:
Generates polynomial from a list of params (coefficients) and also generates noisy (gaussian) polynomials.

```
rv = RandomVariable()

def infer_polynomial(x: int | float | np.ndarray, params: list):
    cexp = 1
    val = 0
    for p in params[::-1]:
        val += p * cexp
        cexp *= x
    return val

def generate_noisy_polynomial(x: int | float | np.ndarray, params: list, mu=0, sigma=20):
    y = infer_polynomial(x, params)
    y = [_y + rv.gaussian(mu=mu, sigma=sigma) for _y in y]
    return np.array(y)
```

Plots of few polynomials with noise.



Curve Fitting:

```python
def get_vander(x, degree):
    vander = []
    for i in range(degree+1):
        vander.append(x**i)
    return np.vstack(vander).T

def fit_polynomial(x, y, degree):
    H = get_vander(x, degree)
    params = np.linalg.inv(H.T @ H) @ H.T @ y
    return params[::-1]

def fit_line(x, y):
    H = np.vstack([np.ones_like(x), x]).T
    params = np.linalg.inv(H.T @ H) @ H.T @ y
    return params[::-1]

def get_fim(x, est_params, est_sigma):
    degree = len(est_params) - 1
    I = np.zeros((degree + 1, degree + 1))

    for i in range(degree + 1):
        for j in range(degree + 1):
            grad_i = x ** i
            grad_j = x ** j
            I[i, j] = np.sum(grad_i * grad_j) / est_sigma**2

    return I
```

Contains code for getting the Vandermonde Matrix (get_vander) and fitting polynomial:

$$H = \begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{vmatrix} \tag{1}$$

$$\theta = (H^T H)^{-1} H^T y = H^\dagger y \tag{2}$$

2.2 Main Code

Line Fitting

```python
import numpy as np
import matplotlib.pyplot as plt
from utils.stochastic import RandomVariable
from utils.polynomial import infer_polynomial, generate_noisy_polynomial, pretty_polynomial
from utils.fit import fit_line, get_fim

rv = RandomVariable()

if __name__ == "__main__":
    # data setup
    x = np.arange(-10, 10, 0.1)
    opt_params =  [4, -5]

    print("Optimal Params: ", pretty_polynomial(opt_params))
    y = generate_noisy_polynomial(x, opt_params, sigma=1) # N(mu=0, sigma=1)

    # estimation
    estimated_params = fit_line(x, y)
    print("Estimated Params: ", pretty_polynomial(estimated_params))

    # fim and covariance
    fim = get_fim(x, estimated_params, 1)
    cov = np.linalg.inv(fim)
    crlb = np.diag(cov)
    print("CRLB:\n", crlb)

    plt.scatter(x, y)
    plt.plot(x, infer_polynomial(x, estimated_params), color="orange", label="estimated")
    plt.show()
```

Polynomial Fitting

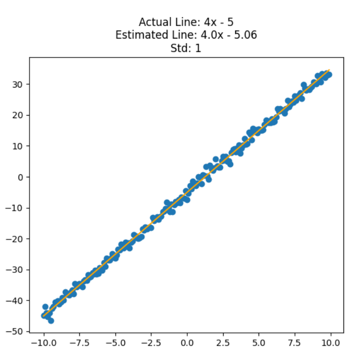
```python
import numpy as np
import matplotlib.pyplot as plt
from utils.stochastic import RandomVariable
from utils.polynomial import infer_polynomial, generate_noisy_polynomial, pretty_polynomial
from utils.fit import fit_polynomial, get_fim

rv = RandomVariable()

if __name__ == "__main__":
    # data setup
    x = np.arange(-10, 10, 0.1)
    degree = 4

    # generating some curve to estimate
    opt_params =  [rv.uniform(-1, 1) for _ in range(degree+1)]
    sigma = 10
    # opt_params = [0.5, 2, 1]
    # degree = len(opt_params)-1

    print("Optimal Params: ", pretty_polynomial(opt_params))
    y = generate_noisy_polynomial(x, opt_params, sigma=sigma) # N(mu=0, sigma=1)

    # estimation
    estimated_params = fit_polynomial(x, y, degree)
    print("Estimated Params: ", pretty_polynomial(estimated_params))

    # fim and covariance
    fim = get_fim(x, estimated_params, 1)
    cov = np.linalg.inv(fim)
    crlb = np.diag(cov)
    print("CRLB:\n", crlb)

    plt.scatter(x, y)
    plt.plot(x, infer_polynomial(x, estimated_params), color="orange", label="estimated")
    plt.title(f"Actual Line: {pretty_polynomial(opt_params)}\nEstimated Line:
{pretty_polynomial(estimated_params)}\nStd: {sigma}")
    plt.show()
```

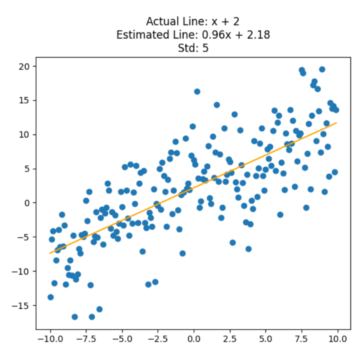
Comparing Mean and Max Estimators

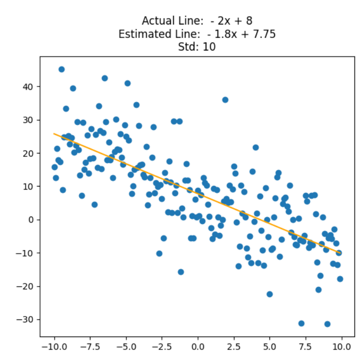
```python
import numpy as np
import matplotlib.pyplot as plt
from utils.stochastic import RandomVariable

def noisy_dc(dc, noise):
    x = np.arange(-10, 10, 0.1)
    y = [dc] * len(x)
    y = [_y + noise() for _y in y]
    return np.array(y)

def dependent_uniform():
    return rv.uniform(-alpha*dc, alpha*dc)

rv = RandomVariable()
n_iters = int(1e4)
dc = 1000
alpha = 1e-2

for noise in [dependent_uniform, rv.gaussian]:
    estimates = {"mean": [], "max": []}

    for _ in range(n_iters):
        y = noisy_dc(dc, noise)
        estimates["mean"].append(np.mean(y))
        estimates["max"].append(np.max(y))

    biases = {"mean": 0, "max": 0}
    stds = {"mean": 0, "max": 0}

    for k in biases.keys():
        biases[k] = np.mean(estimates[k]) - dc
        stds[k] = np.std(estimates[k])

    plt.subplot(1, 2, 1)
    plt.hist(estimates["mean"], label="mean", bins=50)
    plt.title(f"mean | bias:{biases['mean']:.4f}, std:{stds['mean']:.4f}")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.hist(estimates["max"], label="max", bins=50)
    plt.title(f"max | bias:{biases['max']:.4f}, std:{stds['max']:.4f}")
    plt.legend()
    plt.suptitle(str(noise.__name__))
    plt.show()
```

3 Results

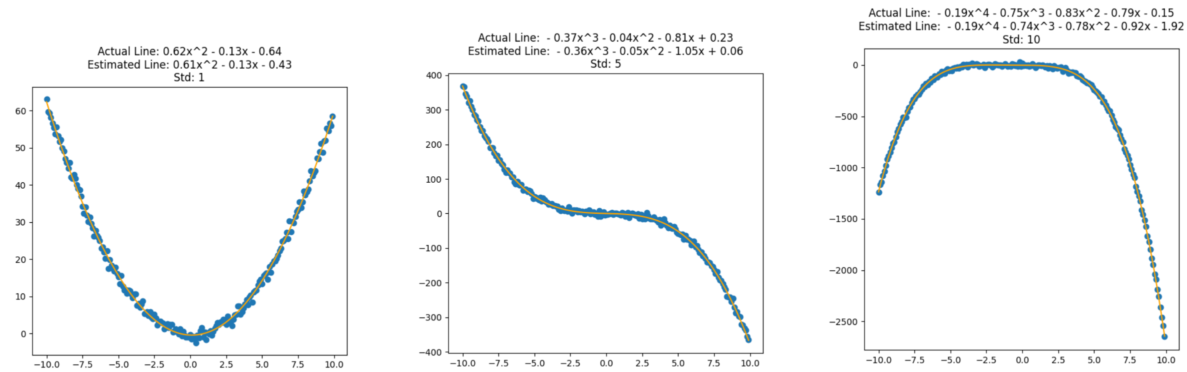
Line Fitting







Polynomial Fitting



Mean vs Max Estimators

Parameter Dependent Uniform - $U(-\alpha * dc, \alpha * dc)$

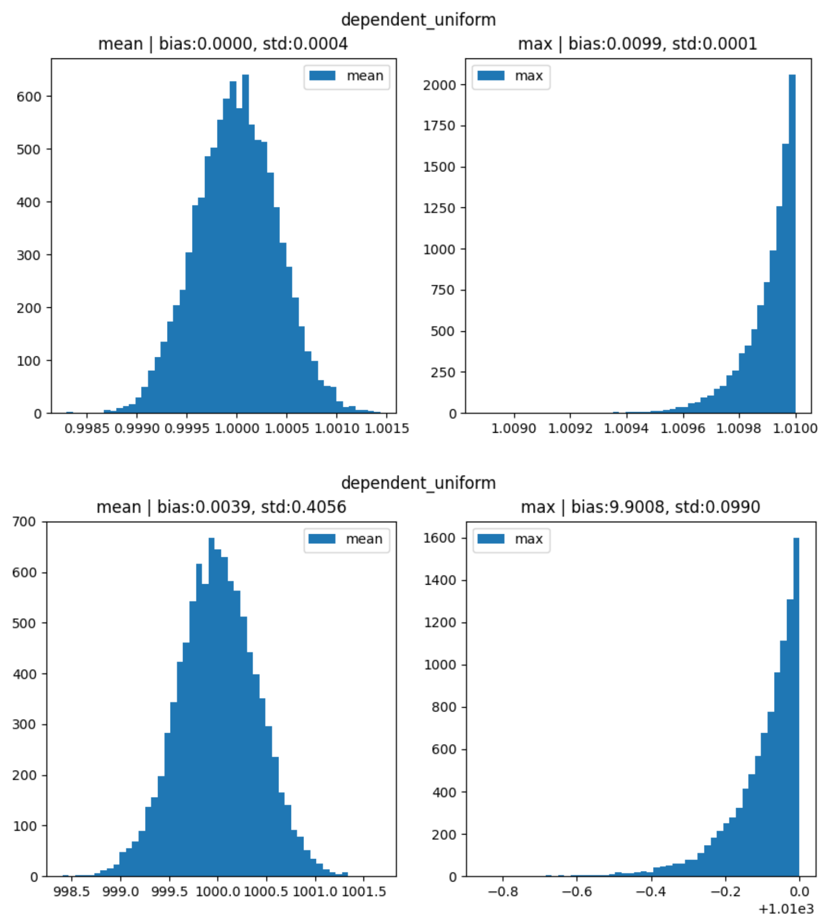


Figure 1: (a) DC=1, (b) DC=1000

Gaussian - $N(0, 1)$

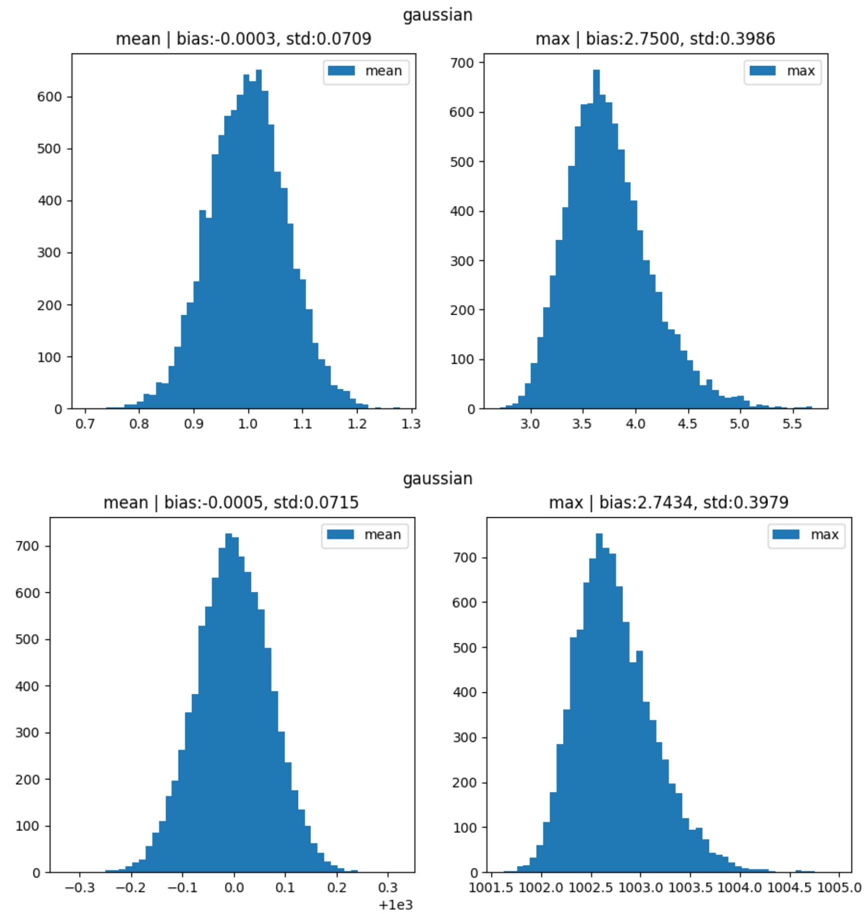


Figure 2: (a) DC=1, (b) DC=1000

4 Discussion

- I generated different types of random variables and was able to verify Law of Large Numbers when I varied number of samples as it more closely approximated the real distribution.
- Using the random variable I had made before I modeled the input data (with gaussian noise)
- Then for the first two parts, the equations for polynomial regression were used to get estimations for the polynomial coefficients

$$H = \begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{vmatrix}$$

$$\theta = (H^T H)^{-1} H^T y = H^\dagger y$$

- The polynomial corresponding to the theta value was then plotted and the Fisher Information Matrix and Cramer Rao Lower Bound was calculated.

$$CRLB = I^{-1}(\theta)$$

- For comparing the mean and max estimators, first a ”dependent” uniform random variable was created:

$$U(-\alpha * DC, \alpha * DC), \alpha = 0.01$$

- It was interesting to see that for the dependent uniform, the mean estimator followed a bell-like curve and the max estimator was similar to a flipped exponential.
- When the DC value was high, the max estimator was more precise in the case of the gaussian.