Assignment-6

January 26, 2023

1 Question 1

[1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
```

[2]:
```
def ols_estimator(y, X, include_constant=True):

    N = X.shape[0]
    if include_constant:
        X_local = np.append(np.ones(N)[None, :].T, X, 1)
    else:
        X_local = np.array(X)

    xt_x_inv = np.linalg.inv(X_local.T@X_local)

    ols_beta = xt_x_inv@(X_local.T@y)
    point_estimates = ols_beta@X_local.T
    residuals = y - point_estimates

    error_vars = residuals*residuals
    error_variance_estimate = np.mean(error_vars)

    ols_variance_matrix = xt_x_inv * error_variance_estimate
    ols_stderr = np.sqrt(np.diag(ols_variance_matrix))

    D_mat = np.diag(error_vars)
    white_variance_matrix = xt_x_inv@(X_local.T@D_mat@X_local)@xt_x_inv
    white_stderr = np.sqrt(np.diag(white_variance_matrix))

    R_2 = 1 - error_variance_estimate/np.var(y)

    max_log_likelyhood = -N/2*(math.log(2*math.pi) + 1) - N/2*math.
↪log(error_variance_estimate)

    P_mat = X_local@xt_x_inv@X_local.T
    M_mat = np.identity(N) - P_mat
```

```
    # (e) Verified that PM = 0
    assert np.allclose(P_mat@M_mat, 0)

    # (f) Verified ex=0
    np.allclose(residuals @ X, 0)

    # (g) Qxx = E(x_i x_i')
    condition_number = np.linalg.cond(X_local.T@X_local) ## Close to singular

    output = {'N': N,
              'betas': ols_beta,
              'point_estimates': point_estimates,
              'residuals': residuals,
              'ols_variance_matrix': ols_variance_matrix,
              'ols_stderr': ols_stderr,
              'white_variance_matrix': white_variance_matrix,
              'white_stderr': white_stderr,
              'R_square': R_2,
              'max_log_likelyhood': max_log_likelyhood,
              'P': P_mat,
              'M': M_mat,
              'condition_number': condition_number
             }


    return output
```

[3]:
```
import scipy

TEST_ALPHAS = [0.01, 0.05, 0.1]

def t_test(b_cap, b_o, stderr, N):
    t_value = np.abs((b_cap - b_o)/stderr)
    reject = {}

    for alpha in TEST_ALPHAS:
        critical_value = scipy.stats.t.ppf(1 - alpha, df=N)
        reject[alpha] = t_value > critical_value
    return t_value, reject
```

[4]:
```
X = np.genfromtxt('X.csv', delimiter=',').T
Y = np.genfromtxt('Y.csv', delimiter=',')
Y.shape, X.shape
```

```
# It looks line the first columns in both X and Y is the index of the value
print((Y[:, 0] == X[:, 0][:-1]).all())
# This shows X has an extra index at the end, which we don't need
X = X[:-1]
Y.shape, X.shape
# Both shapes align, we can now remove the index because it's unnecessary
Y = Y[:, 1:]
X = X[:, 1:]

# We have the X matrix in the required form
# We have two Y values, so we will run ola separately on both of them
y1 = Y[:, 0]
y2 = Y[:, 1]
```

```
True
```

[5]:
```
def print_output(output):
    print(f"Betas: {output['betas']}")
    print(f"OLS Standard Errors: {output['ols_stderr']}")
    print(f"White Standard Errors: {output['white_stderr']}")
    print(f"R Square: {output['R_square']}")
    print(f"Maximum Log Likelyhood: {output['max_log_likelyhood']}")
```

[6]:
```
print("\n**Running OLS on y column 1**\n")

output1 = ols_estimator(y1, X)
print_output(output1)

print("\n**Running OLS on y column 2**\n")

output2 = ols_estimator(y2, X)
print_output(output2)
```

```
**Running OLS on y column 1**

Betas: [ 0.36075732  1.02258321 -0.14831037 -0.26284836]
OLS Standard Errors: [0.03552488 0.00711278 0.01164566 0.01040885]
White Standard Errors: [0.03519608 0.00941046 0.02282585 0.01586761]
R Square: 0.9514861152178427
Maximum Log Likelyhood: -1750.221752833414

**Running OLS on y column 2**

Betas: [-0.1765854   1.24368112 -0.17476801  1.05424915]
OLS Standard Errors: [0.17065781 0.03416903 0.05594452 0.05000306]
White Standard Errors: [0.1950188  0.05487085 0.07937635 0.07605262]
```

```
R Square: 0.6770470562833228
Maximum Log Likelyhood: -3492.2859741193697
```

[7]:
```
def print_test_output(t_output, value):
    print(f"t values: {t_output[0]}")
    for alpha in t_output[1]:
        out = t_output[1][alpha]
        print(f"Test beta={value} rejected at alpha={alpha*100:2.0f}%: {out}")

print("\n**Running T-test on y column 1**\n")

print_test_output(t_test(output1['betas'], 0, output1['white_stderr'],␣
 ↪output1['N']), 0)
print_test_output(t_test(output1['betas'], 1, output1['white_stderr'],␣
 ↪output1['N']), 1)

print("\n**Running OLS on y column 2**\n")
print_test_output(t_test(output2['betas'], 0, output2['white_stderr'],␣
 ↪output2['N']), 0)
print_test_output(t_test(output2['betas'], 1, output2['white_stderr'],␣
 ↪output2['N']), 1)
```

Running T-test on y column 1

```
t values: [ 10.24992903 108.6645193    6.49747309  16.56508699]
Test beta=0 rejected at alpha= 1%: [ True  True  True  True]
Test beta=0 rejected at alpha= 5%: [ True  True  True  True]
Test beta=0 rejected at alpha=10%: [ True  True  True  True]
t values: [18.16232596  2.3997984  50.30744339 79.58654568]
Test beta=1 rejected at alpha= 1%: [ True  True  True  True]
Test beta=1 rejected at alpha= 5%: [ True  True  True  True]
Test beta=1 rejected at alpha=10%: [ True  True  True  True]
```

Running OLS on y column 2

```
t values: [ 0.90547884 22.66560561  2.20176412 13.86210083]
Test beta=0 rejected at alpha= 1%: [False  True False  True]
Test beta=0 rejected at alpha= 5%: [False  True  True  True]
Test beta=0 rejected at alpha=10%: [False  True  True  True]
t values: [ 6.03318965  4.44099384 14.79997452  0.71331072]
Test beta=1 rejected at alpha= 1%: [ True  True  True False]
Test beta=1 rejected at alpha= 5%: [ True  True  True False]
Test beta=1 rejected at alpha=10%: [ True  True  True False]
```

2 Question 2

(a)

$$y_i = \alpha + \beta x_i + e_i$$
$$y_i = (\alpha, \beta) * (1, x_i)' + e_i$$

$\hat{\alpha}$ and $\hat{\beta}$ can be said to be unbiased and consistent since this is the OLS estimator and x_i and e_i are independent.

Since we know both x_i and e_i are normally distributed, we can say $(\hat{\alpha}, \hat{\beta})$ are also normally distributed.

$$\hat{\beta} = Cov(x_i, y_i)/Var(x_i)$$
$$\hat{\alpha} = E(y_i) - E(x_i) * \hat{\beta}$$

$$E(\hat{\alpha}) = \alpha$$
$$E(\hat{\beta}) = \beta$$

$$Var(e_i) = \sigma^2$$
$$E(Var(\hat{e_i})) = \sigma^2$$

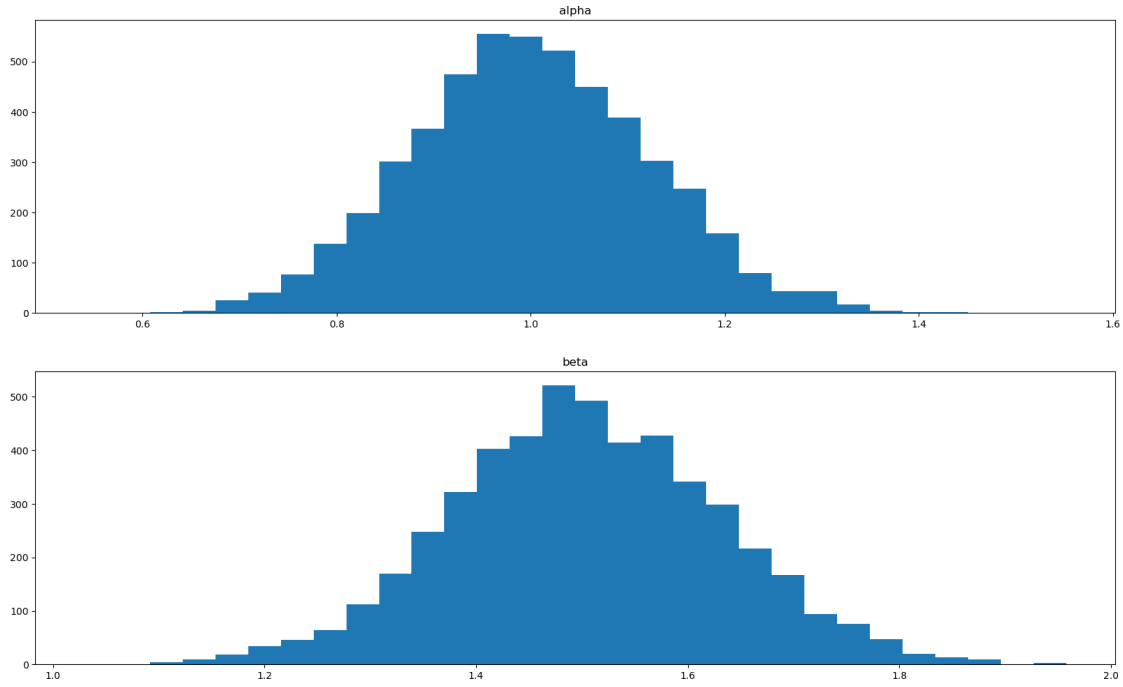
$$Var(\hat{\alpha}) = \frac{(\bar{X})^2\sigma^2}{nVar(x)} + \frac{\sigma^2}{n}$$
$$Var(\hat{\beta}) = \frac{\sigma^2}{nVar(x)}$$

[10]:
```
M = 5000
N = 50

X_mn = np.random.normal(size=(M, N))
E_mn = np.random.normal(size=(M, N))
```

[20]:
```
def Q2_distribution_analysis(X_mn, E_mn, M, N):

    Y_mn = X_mn + E_mn + 1

    estimated_betas = []
    estimated_alphas = []

    for m in range(M):
        X_n = X_mn[m]
        Y_n = Y_mn[m]
        E_n = E_mn[m]
```

```
        cov_matrix = np.cov(X_n, Y_n)
        estimated_beta = cov_matrix[0, 1] / cov_matrix[0, 0]
        estimated_alpha = np.mean(Y_n) - np.mean(X_n) * estimated_beta

        estimated_betas.append(estimated_beta)
        estimated_alphas.append(estimated_alpha)

    estimated_betas = np.array(estimated_betas)
    estimated_alphas = np.array(estimated_alphas)

    mean_beta = np.mean(estimated_betas)
    mean_alpha = np.mean(estimated_alphas)

    std_beta = np.std(estimated_betas)
    std_alpha = np.std(estimated_alphas)

    print(f"Mean alpha, beta = {mean_alpha, mean_beta}")
    print(f"Std alpha, beta = {std_alpha, std_beta}")

    fig, ax = plt.subplots(2, figsize=(20,12))
    ax[0].hist(estimated_alphas, bins=30)
    ax[0].set_title(f"alpha")

    ax[1].hist(estimated_betas, bins=30)
    ax[1].set_title(f"beta")

    fig.show()
    return

Q2_distribution_analysis(X_mn, E_mn, M, N)
```

```
Mean alpha, beta = (1.000624252223831, 1.5041020341767508)
Std alpha, beta = (0.12221743941603834, 0.12699293873397874)
```



(d) The experimental results match the theoretical analysis, which should give:

Mean alpha, beta = (1, 1)

Std alpha, beta = (sqrt(0.02) = 0.141, sqrt(0.02) = 0.141)

and both alpha and beta are normally distributed

3 Question 3

(a, d) Since X and E are correlated, we can no longer use the same formulas. alpha and beta would still be normally distributed.

Beta would be higher, because the numerator would be cov(x, y) = cov(x, x) + cov(x, e) = var(x) + 0.5 So, beta is 1.5 theoretically.

Alpha will still be 1. Both expected value of y and beta will increase proportionally, keeping the alpha same.

Though variance of both alpha and beta should be lower because of the correlation between x and y. I haven't derived the exact formula.

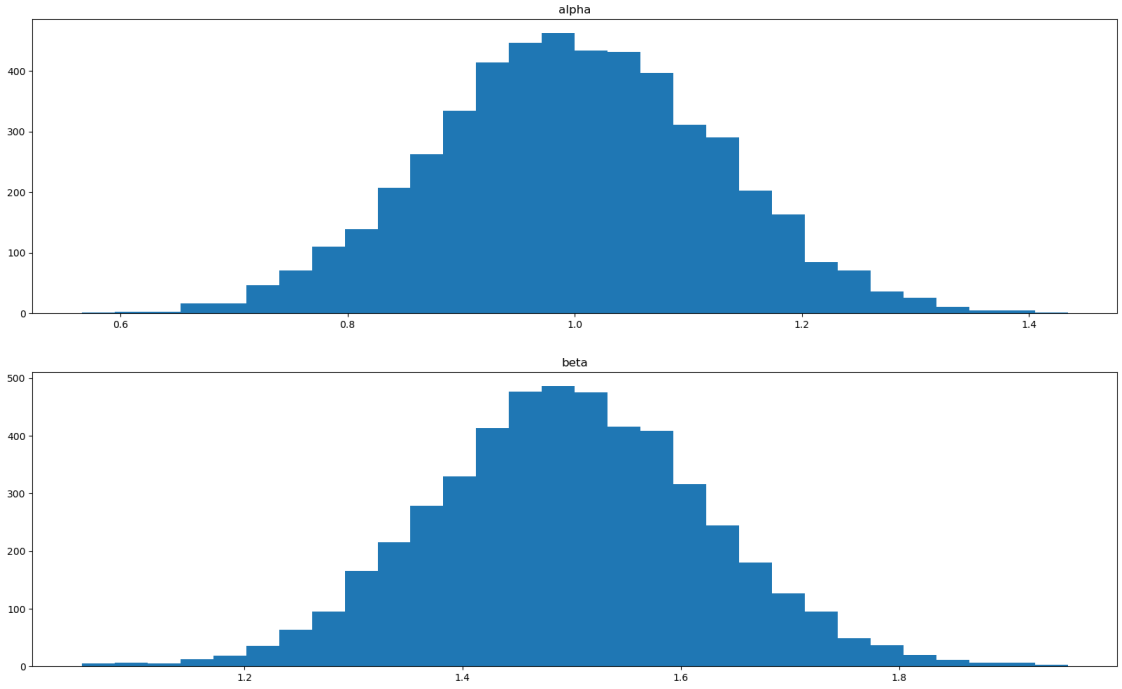
Below experiment gives results in line with the theoretical expectations.

[21]:
```
A1_mn = np.random.normal(size=(M, N))
A2_mn = np.random.normal(size=(M, N))
```

```
corr = 0.5

X_mn = A1_mn
E_mn = corr*A1_mn + math.sqrt(1-corr*corr)* A2_mn
```

[22]:
```
Q2_distribution_analysis(X_mn, E_mn, M, N)
```

```
Mean alpha, beta = (0.9996738981875842, 1.4974343414927882)
Std alpha, beta = (0.1236087312970405, 0.12624029246825924)
```



4 Question 4

(a, d) Similar to Q2, x and e are linearly independent. So we can use the same results.

$\hat{\alpha}$ and $\hat{\beta}$ are unbiased and consistent since this is the OLS estimator and x_i and e_i are independent.

$$\hat{\beta} = Cov(x_i, y_i)/Var(x_i)$$
$$\hat{\alpha} = E(y_i) - E(x_i) * \hat{\beta}$$

$$E(\hat{\alpha}) = \alpha$$
$$E(\hat{\beta}) = \beta$$

$$\hat{\beta} = 1/1 = 1$$

$$\hat{\alpha} = 1 - 0 * 1 = 1$$

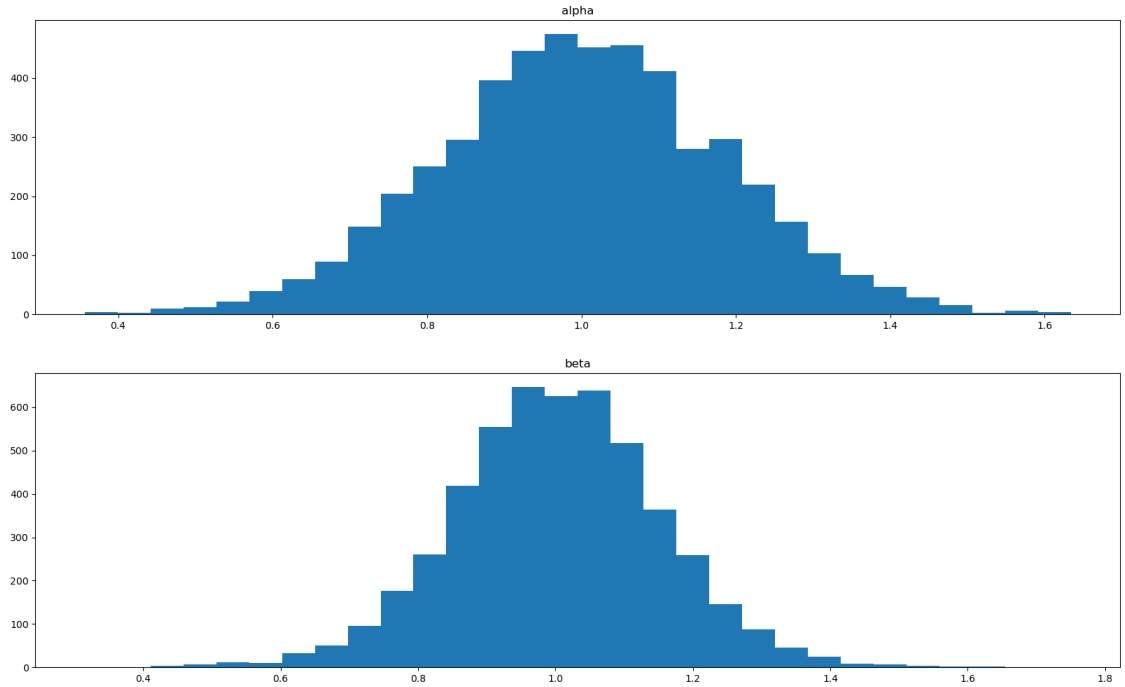
Since x_i and e_i have a t-distribution, it is hard to derive the formula for the variance. But we can judge that the variance would be higher than normal distribution, because the distribution is fat-tailed relative to the normal distribution.

Below experiment gives results in line with the theoretical expectations.

[27]:
```
X_mn = np.random.standard_t(5, size=(M, N))
E_mn = np.random.standard_t(5, size=(M, N))
```

[28]:
```
Q2_distribution_analysis(X_mn, E_mn, M, N)
```

```
Mean alpha, beta = (1.0008510280545677, 1.002312917398825)
Std alpha, beta = (0.1836347766624594, 0.1505610830078113)
```



5 Question 5

(a, d) Similar to Q2, x and e are linearly independent. So we can use the same results.

$\hat{\alpha}$ and $\hat{\beta}$ are unbiased and consistent since this is the OLS estimator and x_i and e_i are independent.

$$\hat{\beta} = Cov(x_i, y_i)/Var(x_i)$$
$$\hat{\alpha} = E(y_i) - E(x_i) * \hat{\beta}$$

$$E(\hat{\alpha}) = \alpha$$
$$E(\hat{\beta}) = \beta$$

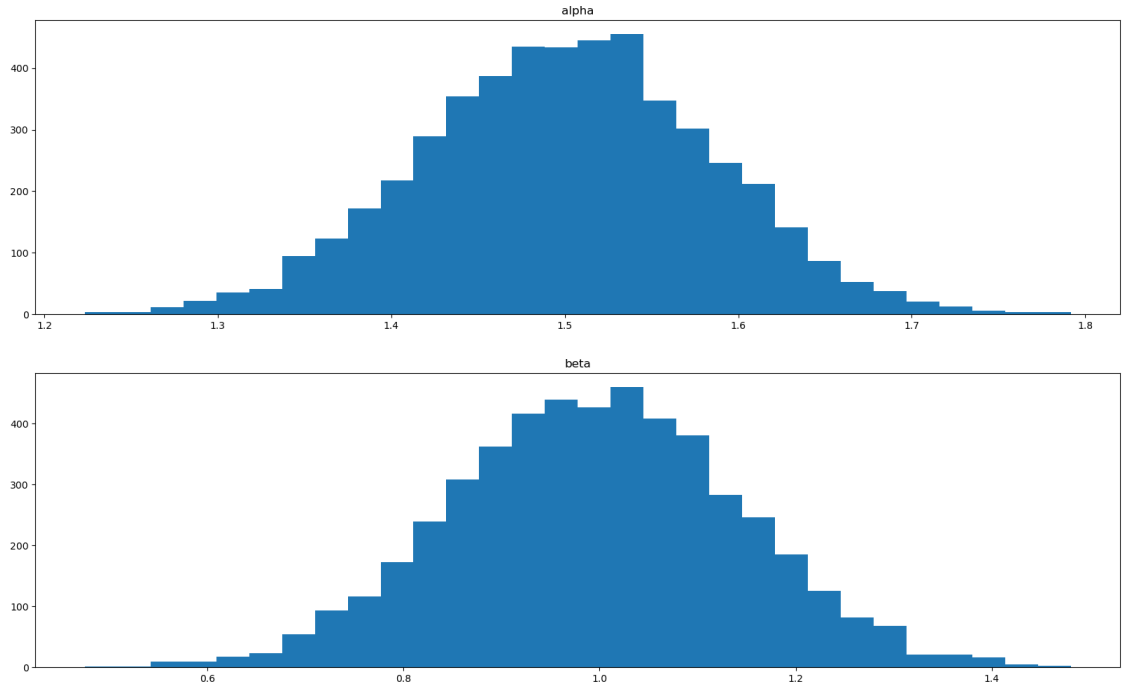
$$\hat{\beta} = (1/4)/(1/4) = 1$$
$$\hat{\alpha} = 2 - 1 * 0.5 = 1.5$$

Since x_i and e_i have a uniform distribution, it is hard to derive the formula for the variance. Below experiment gives results in line with the theoretical expectations.

[25]:
```
X_mn = np.random.uniform(size=(M, N))
E_mn = np.random.uniform(size=(M, N))
```

[26]:
```
Q2_distribution_analysis(X_mn, E_mn, M, N)
```

```
Mean alpha, beta = (1.5012993409286401, 0.9960356058230478)
Std alpha, beta = (0.08375138757877987, 0.14552584965710202)
```



6 Question 6

1. As N increases, the estimated values of alpha and betas become closer to the real values.
2. As N increases, the variance becomes tighter for all Q2-Q5, because the more information in the system, the more we would be sure about the estimates. (The formula of variance has N in denominator)

[ ]: