Fintech545 Project Week2

Kaiye Chen

September 2022

1 Problem1

1.1 Mathematical Provement

Given X fits normal distribution:

$$X \sim N(\mu, \Sigma) \tag{1}$$

Based on the partition matrix:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times 1 \\ (N-q) \times 1 \end{bmatrix}$$

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times 1 \\ (N-q) \times 1 \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times q & q \times (N-q) \\ (N-q) \times q & (N-q) \times (N-q) \end{bmatrix}$$

Then given the condition:

$$X_2 = a \tag{2}$$

Combining the matrix with (2), we can have:

$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(a - \mu_2)$$

$$\bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{12}$$

Then we turn to OLS. We have:

$$Y = X\beta$$

Multiply each side by $(X'X)^{-}1X'$. Considering the matrix form of X and Y:

$$\cdot X = [1 \quad x] = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ ... & ... \\ 1 & x_n \end{bmatrix}, Y = y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix}, \text{and } \beta = [\beta_0 \quad \beta_1]$$

We can have:

$$\hat{\beta_0} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\hat{\beta_1} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Based on the unbiased estimation of the variance term value σ, we can have:

$$\widehat{\beta}_0 = \overline{y}$$

$$\widehat{\beta}_1 = \frac{\sigma_{xy}}{\sigma_x^2}$$

Then we used the original equation of linear function:

$$\hat{y} = \hat{\beta_0} + \hat{\beta_1}x$$

$$\hat{y} - \bar{y} = \hat{\beta_0} + \hat{\beta_1}x - (\hat{\beta_0} + \hat{\beta_1}\bar{x})$$

$$\hat{y} = \bar{y} + \hat{\beta_1}(x - \bar{x})$$

We can get the function that:

$$\mu = \mu_y + \frac{\sigma_{xy}}{\sigma_x^2}(x - \mu_x)$$

Also, noticed that $\sigma_{xy} = \Sigma_{12}$, $\sigma_x^2 = \Sigma_{22}$: This equation for the mean is the same as the final equation of multivariate normal distribution. Equation for variance is similar.

1.2 Empirical Provement

Based on the equations above, we first go with multivariate distribution:

In [10]:
```
mean_x = data['x'].mean()
mean_y = data['y'].mean()
print(f"mu_x = {mean_x}, mu_y = {mean_y}")
```

```
mu_x = -0.14054562796809716, mu_y = -0.022277188035264378
```

In [11]:
```
cov = data.cov()
print(cov)
variance_x = cov.loc["x","x"]
variance_y = cov.loc["y","y"]
covariance_xy = cov.loc["x","y"]
```

```
          x         y
x  1.315195  0.562908
y  0.562908  0.898883
```

In [23]:
```
con_mean = mean_y + (covariance_xy/variance_x)*(data.x - mean_x)
con_variance = variance_y - covariance_xy**2/variance_x
```

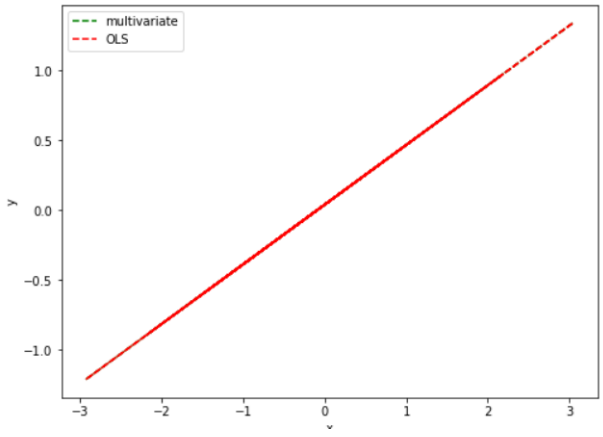
Then it comes out with OLS:

In [14]:
```
results1 = stat.OLS(data.y, stat.add_constant(data.x)).fit()
ols_mean_y = results1.fittedvalues
ols_variance_y = results1.ssr / (len(data.x)-2)
```

At this stage, we have two series of conditional means for multivariate and OLS, in the meantime, two variances.

:
```
figure, comp = plt.subplots(figsize=(8,6))
comp.plot(data.x, con_mean, 'g--', label="multivariate")
comp.plot(data.x, results1.fittedvalues, 'r--', label="OLS")
comp.set_xlabel('x')
comp.set_ylabel('y')
comp.legend(loc='best')
```

:
```
<matplotlib.legend.Legend at 0x22d087d62b0>
```



:
```
print(con_variance)
print(ols_variance_y)
```

```
0.6579563030192094
0.6646701428459357
```

You can see that in the plot, two lines which reflects the conditional means of two method are merged in one. Then, the value of variance is similar, but not totally the same.

2 Problem2

2.1 How well does it fit the assumption of normally distributed errors?

We can still use the package above and also the same method to calculate the error term. The error term is derived from the difference between fitted y and actual y.

```
In [26]: import scipy.stats as st
         import scipy.optimize as opt

In [25]: data1 = pd.read_csv("problem2.csv")
         results = stat.OLS(data1.y, stat.add_constant(data1.x)).fit()
         error = data1.y - results.fittedvalues

         C:\Users\51069\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:1
         f concat except for the argument 'objs' will be keyword-only
           x = pd.concat(x[::order], 1)

In [27]: s,p = st.shapiro(error)
         alpha = 0.1
         print(f"t-test statistic = {s}, p = {p}")
         if p < alpha:
             print("Reject H0.")
         else:
             print("Can't reject H0")

         t-test statistic = 0.938385546207428, p = 0.00015389148029498756
         Reject H0.
```

We can do a t-test(statistically, but here for the small sample the accurate name is Shapiro-Wilk test) to understand the normality of error term distribution. The result shows that we need to reject the hypothesis that error term is normally distributed.

2.2 Compare T and normal distribution with MLE method

We first calculate the likelihood function of two kinds of distribution, then optimize it.

In [72]:
```
from scipy.stats import norm
from scipy.stats import t
import scipy.optimize as optimize
```

In [83]:
```
def log_like_nor(par, y, x):
    e = y - par[0] - par[1]*x
    likelihood = -np.log(norm(0,par[2]).pdf(e)).sum()
    return likelihood
def log_like_t(par, y, x):
    e = y - par[0] - par[1]*x
    likelihood = -np.log(t(df = par[2],scale = par[3]).pdf(e)).sum()
    return likelihood
```

In [84]:
```
opt_res_norm = optimize.minimize(fun = log_like_nor,
                                 x0 = [0, 0, 1],
                                 args = (data1.y, data1.x))
opt_res_t = optimize.minimize(fun = log_like_t,
                                 x0 = [0, 0, 10, 1],
                                 args = (data1.y, data1.x))
```

Then we define different functions to calculate the metrics to compare MLE results under 2 different distribution.

```
def cal_SSE(opt_res_t, opt_res_norm, data):
    e_t = data.y - opt_res_t.x[0] - opt_res_t.x[1]*data.x
    e_norm = data.y - opt_res_norm.x[0] - opt_res_norm.x[1]*data.x
    SSE_t = sum(e_t*e_t)
    SSE_norm = sum(e_norm*e_norm)
    return SSE_norm,SSE_t
```

```
def cal_AIC(opt_res_t, opt_res_norm, data):
    AIC_t = 2*4 + 2*opt_res_t.fun
    AIC_norm = 2*3 + 2*opt_res_norm.fun
    return AIC_norm, AIC_t
```

```
def cal_BIC(opt_res_t, opt_res_norm, data):
    BIC_t = 4*np.log(len(data.y)) + 2*opt_res_t.fun
    BIC_norm = 3*np.log(len(data.y)) + 2*opt_res_norm.fun
    return BIC_norm, BIC_t
```

```
SSE_norm,SSE_t = cal_SSE(opt_res_t, opt_res_norm, data1)
AIC_norm,AIC_t = cal_AIC(opt_res_t, opt_res_norm, data1)
BIC_norm,BIC_t = cal_BIC(opt_res_t, opt_res_norm, data1)
print("Distribution,b0,b1,SSE,AIC,BIC")
print(f"Norm, {opt_res_norm.x[0]}, {opt_res_norm.x[1]}, {SSE_norm}, {AIC_norm}, {BIC_norm} ")
print(f"t, {opt_res_t.x[0]}, {opt_res_t.x[1]}, {SSE_t}, {AIC_t}, {BIC_t} ")
```

```
Distribution,b0,b1,SSE,AIC,BIC
Norm, 0.11983620432704001, 0.6052048467660482, 143.61484854062613, 325.9841933783247, 333.79970393628895
t, 0.14261408167008574, 0.5575717648546195, 143.88176185026407, 318.94594082493296, 329.3666215688853
```

In the comparing result above, we can see that the coefficient and intercept are all different. So we can say that the breaking of the normality assumption will lead to different expected values. In order to make sure which distribution fits better, we can see that only the SSE, which only reflect the accuracy, normal distribution have a better results. All the other two metrics, which combine accuracy and low complexity, all indicates that t distribution is better. So we can say that t distribution fits better overall.

3 Problem3

To simulate this process, we define two very similar functions, which automatically generate ACF and PACF plot with diffrent lag terms.

9]:
```
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.graphics.tsaplots import plot_pacf,plot_acf
```

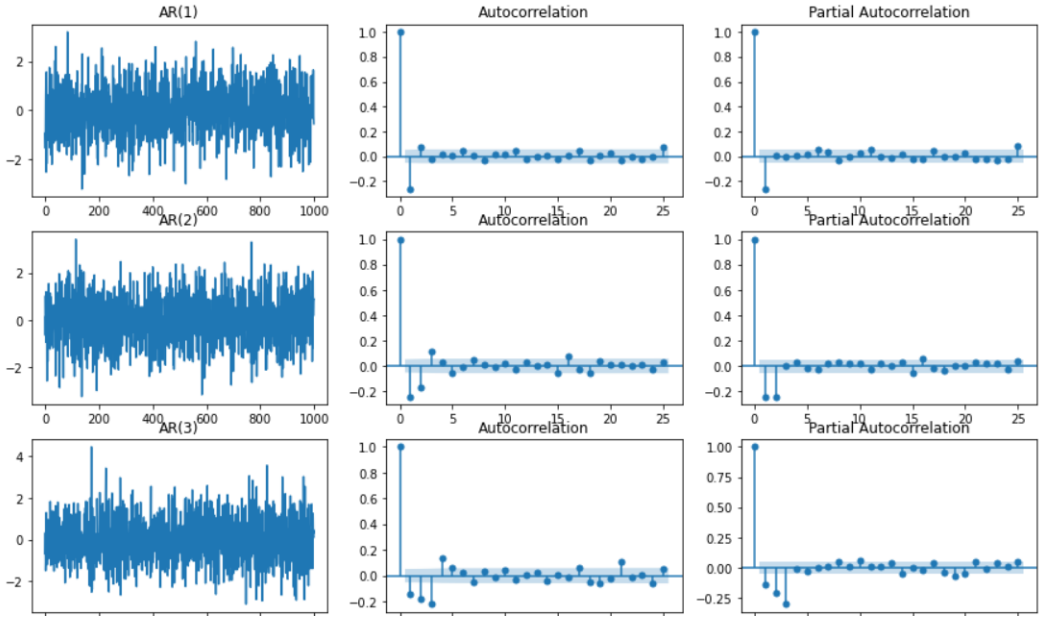
0]:
```
def AR_figures(params,titles):
    length = len(params)
    fig, axes = plt.subplots(length,3,figsize = (5*length,3*length))
    for i in range(len(params)):
        process = ArmaProcess(ar = params[i])
        simulated_data = process.generate_sample(nsample=1000)
        axes[i][0].plot(simulated_data)
        axes[i][0].set_title(titles[i])
        fig = plot_acf(simulated_data, alpha=0.1, lags=25, ax=axes[i][1])
        fig = plot_pacf(simulated_data, alpha=0.1, lags=25, ax=axes[i][2])

def MA_figures(params,titles):
    length = len(params)
    fig, axes = plt.subplots(length,3,figsize = (5*length,3*length))
    for i in range(len(params)):
        process = ArmaProcess(ma = params[i])
        simulated_data = process.generate_sample(nsample=1000)
        axes[i][0].plot(simulated_data)
        axes[i][0].set_title(titles[i])
        fig = plot_acf(simulated_data, alpha=0.1, lags=25, ax=axes[i][1])
        fig = plot_pacf(simulated_data, alpha=0.1, lags=25, ax=axes[i][2])
```

Then, with the lag term coefficient of constant 0.25, we generate graphs below to show AR and MA process through different order.

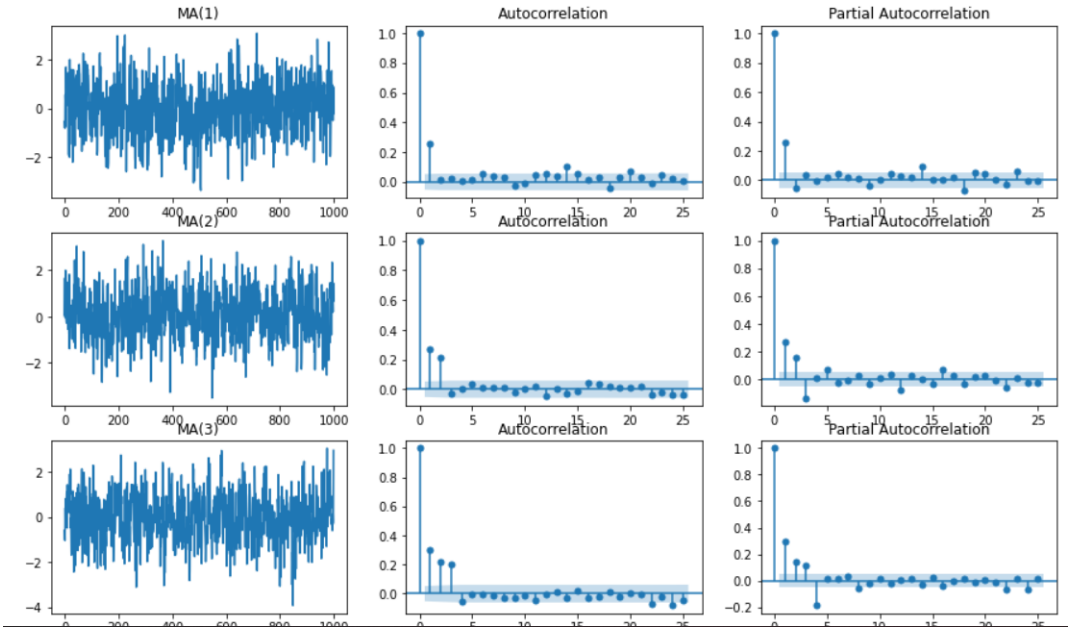
```
AR_coefs = [[1, 0.25], [1, 0.25, 0.25], [1, 0.25, 0.25, 0.25]]
AR_titles = ["AR(1)", "AR(2)", "AR(3)"]

AR_figures(AR_coefs, AR_titles)
```



```
MA_coefs = [[1, 0.25], [1, 0.25, 0.25], [1, 0.25, 0.25, 0.25]]
MA_titles = ["MA(1)","MA(2)","MA(3)"]

MA_figures(MA_coefs, MA_titles)
```



We can see that no matter in MA or AR process, the autocorrelation and partial autocorrelation will all decay to near zero after the lag terms equal to its order.