Finmath 36702, Portfolio Credit Loss, Assignment 1

Joshua Weekes

Package Imports

In [ ]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

Problem 1

In [ ]:
```
np.random.seed(99)

samples = 10000
record_size = 20
variables = 10

rho = 0.3

cov_matrix = np.eye(variables)
cov_matrix[:variables-1, variables-1] = rho
cov_matrix[variables-1, :variables-1] = rho

decomposed = np.linalg.cholesky(cov_matrix)

ind_normals = np.random.randn(samples, record_size, variables)

correlated_normals = ind_normals @ decomposed.T

corr_normals_swapped = np.swapaxes(correlated_normals, 1, 2)
```

In [ ]:
```
betas = np.zeros((variables-1, samples))
r_squared = np.zeros((variables-1, samples))
mse = np.zeros((variables-1, samples))
null_model_mse = np.zeros((samples))

for i in range(samples):
    y = corr_normals_swapped[i, -1, :]
    null_model_mse[i] = ((y - y.mean()) ** 2).mean()

    for j in range(variables-1):
        x = corr_normals_swapped[i, j, :]
        # x = sm.add_constant(x)
        model = sm.OLS(y, x)
```

```
        betas[j, i] = model.fit().params[0]
        r_squared[j, i] = model.fit().rsquared
        predictions = model.fit().predict(x)
        mse[j, i] = ((predictions - y) ** 2).mean()
```

In [ ]:
```
labels = [f'X_{i}' for i in range(1, variables)]
labels_betas = [f'Beta_{i}' for i in range(1, variables)]
labels_r_squared = [f'R^2_{i}' for i in range(1, variables)]
labels_mse = [f'MSE_{i}' for i in range(1, variables)]
```

In [ ]:
```
betas_df = pd.DataFrame(betas.T, columns=labels_betas)
r_squared_df = pd.DataFrame(r_squared.T, columns=labels_r_squared)
mse_df = pd.DataFrame(mse.T, columns=labels_mse)
```

In [ ]:
```
def get_max_r2_beta(row, max_index=10):
    r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
    max_r2_index = np.argmax(r2_values)
    beta_value = row[f'Beta_{max_r2_index + 1}']

    return beta_value
```

In [ ]:
```
combined_df = pd.concat([betas_df, r_squared_df], axis=1)

combined_df.loc[:, 'Proc_1'] = betas_df.loc[:, 'Beta_1']
combined_df.loc[:, 'Proc_2'] = combined_df.apply(get_max_r2_beta, args=[3], axis=1)
combined_df.loc[:, 'Proc_3'] = combined_df.apply(get_max_r2_beta, axis=1)
```

In [ ]:
```
procedure_labels = [f'Procedure {i}' for i in range(1, 4)]
procedure_labels

avg_slope = []
est_bias = []
for i in range(3):
    avg_slope.append(combined_df.loc[:, f'Proc_{i + 1}'].mean())
    est_bias.append(avg_slope[i] - 0.3)

question_1_results = pd.DataFrame({'Average Slope': avg_slope, 'Estimated Bias': es
                       index=procedure_labels)

pd.set_option('display.float_format', '{:.4f}'.format)
question_1_results
```

Out[ ]:

	Average Slope	Estimated Bias
Procedure 1	0.3021	0.0021
Procedure 2	0.4094	0.1094
Procedure 3	0.5678	0.2678

Problem 2

In [ ]:
```
def get_max_r2_mse(row, max_index=10):
    r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
    max_r2_index = np.argmax(r2_values)
    mse = row[f'MSE_{max_r2_index + 1}']

    return mse
```

In [ ]:
```
mse_combined_df = pd.concat([mse_df, r_squared_df], axis=1)

mse_combined_df.loc[:, 'Proc_0'] = null_model_mse
mse_combined_df.loc[:, 'Proc_1'] = mse_df.loc[:, 'MSE_1']
mse_combined_df.loc[:, 'Proc_2'] = mse_combined_df.apply(get_max_r2_mse, args=[3],
mse_combined_df.loc[:, 'Proc_3'] = mse_combined_df.apply(get_max_r2_mse, axis=1)
```

In [ ]:
```
procedure_labels_q2 = [f'Procedure {i}' for i in range(4)]
procedure_labels_q2

avg_mse = []
for i in range(4):
    avg_mse.append(mse_combined_df.loc[:, f'Proc_{i}'].mean())

question_2_results = pd.DataFrame({'Avereage ESE': avg_mse},
                                  index=procedure_labels_q2)

pd.set_option('display.float_format', '{:.3e}'.format)
question_2_results
```

Out[ ]:

	Avereage ESE
Procedure 0	9.556e-01
Procedure 1	8.681e-01
Procedure 2	7.992e-01
Procedure 3	6.473e-01

Not sure why I'm getting that procedure 3 is the best.

Problem 3

In [ ]:
```
def simulate_regressions_rho_list(rho_values, n_samples=1000, sample_size=25):
    results = []

    for rho in rho_values:
        # Set up the covariance matrix based on rho
        cov_matrix = np.array([[1, rho], [rho, 1]])

        # Variables to keep track of slope coefficients and significance
        slopes = []
        significant_slopes = []
        significant_count = 0
```

```
        for _ in range(n_samples):
            # Generate bivariate normal distributions
            x, y = np.random.multivariate_normal([0, 0], cov_matrix, size=sample_si

            # Adding constant to X for OLS regression
            X_with_const = sm.add_constant(x)
            model = sm.OLS(y, X_with_const).fit()
            slope_coeff = model.params[1]
            p_value = model.pvalues[1]

            slopes.append(slope_coeff)
            if p_value < 0.05:
                significant_count += 1
                significant_slopes.append(slope_coeff)

        # Calculate bias of significant slopes (if any) and fraction of significant
        avg_slope = np.mean(slopes)
        bias = (np.mean(significant_slopes - rho)) if significant_count else None
        fraction_significant = significant_count / n_samples

        results.append({
            'rho': rho,
            'bias_of_significant_slopes': bias,
            'fraction_significant': fraction_significant,
            'avg_slope': avg_slope
        })

    return pd.DataFrame(results)
```

In [ ]:

```
rho_values = np.arange(0, 1, 0.1)
pd.set_option('display.float_format', '{:.4f}'.format)
df_results_question3 = simulate_regressions_rho_list(rho_values, n_samples=10000)
df_results_question3
```

Out[ ]:

	rho	bias_of_significant_slopes	fraction_significant	avg_slope
0	0.0000	-0.0183	0.0535	-0.0035
1	0.1000	0.2981	0.0744	0.0978
2	0.2000	0.2809	0.1664	0.2023
3	0.3000	0.2065	0.3174	0.3027
4	0.4000	0.1355	0.5263	0.4003
5	0.5000	0.0714	0.7506	0.5002
6	0.6000	0.0249	0.9135	0.5980
7	0.7000	0.0057	0.9842	0.6998
8	0.8000	-0.0018	0.9994	0.7979
9	0.9000	-0.0005	1.0000	0.8995

Problem 4

In [ ]:
```
def calculate_ese(rho, sample_size, n_samples=10000):
    ese_null = []
    ese_regression = []

    for _ in range(n_samples):
        cov_matrix = np.array([[1, rho], [rho, 1]])
        x, y = np.random.multivariate_normal([0, 0], cov_matrix, size=sample_size).

        null_forecast = np.mean(y)
        ese_null_sample = np.mean((y - null_forecast) ** 2)

        X_with_const = sm.add_constant(x)
        model = sm.OLS(y, X_with_const).fit()
        intercept = model.params[0]
        slope_coeff = model.params[1]

        regression_forecast = intercept + slope_coeff * x
        ese_regression_sample = np.mean((y - regression_forecast) ** 2)

        ese_null.append(ese_null_sample)
        ese_regression.append(ese_regression_sample)

    avg_ese_null = np.mean(ese_null)
    avg_ese_regression = np.mean(ese_regression)

    return avg_ese_null, avg_ese_regression
```

In [ ]:
```
n_values = np.arange(10, 60, 10)
results = []
for N in n_values:
    avg_ese_null, avg_ese_regression = calculate_ese(rho, N)
    results.append((N, avg_ese_null, avg_ese_regression))

df_ese = pd.DataFrame(results, columns=['N', 'Average ESE Null', 'Average ESE Regre
df_ese
```

Out[ ]:

	N	Average ESE Null	Average ESE Regression
0	10	0.8956	0.7272
1	20	0.9499	0.8192
2	30	0.9690	0.8508
3	40	0.9743	0.8646
4	50	0.9823	0.8768

I would expect the error to get better as sample size increases but it seems to get worse. Not entirely sure why this is the case.